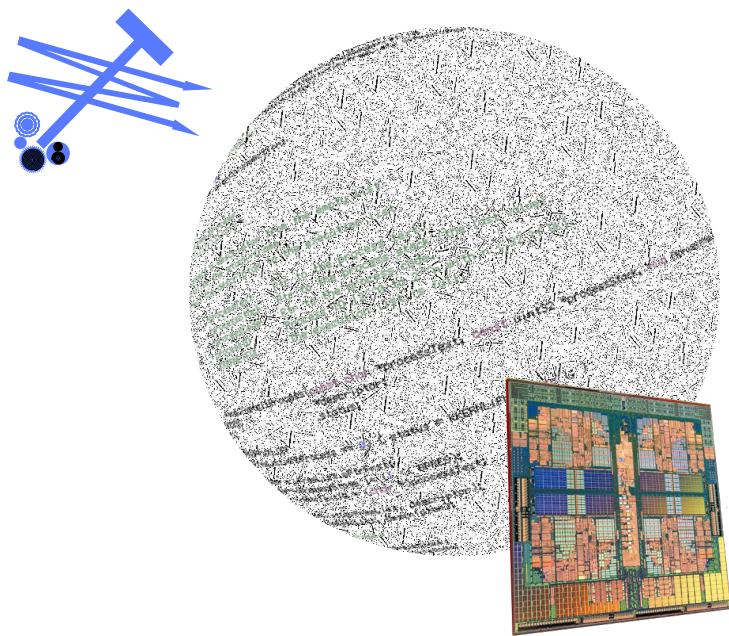


μKOS-III

An RTOS for embedded systems

ARM Cortex M3-M4-M7 and CSEM-IcyCAM SoC



Author:

Edo. Franzi

edo.franzi@ukos.ch

<http://www.ukos.ch>

μKOS-III

An RTOS for embedded systems

ARM Cortex M3-M4-M7 and CSEM-IcyCAM SoC

Preliminary project report V. 4.8.0

the 20th of August, 2017

μKOS project. Copyright 1992-2017, Edo. Franzi.

Sysquake documentation included with permission. Copyright 1997-2017, Calerga Sarl.

*... to all people who cannot sleep at
night, and particularly, to those who
have shared my motivation ...*

1. Introduction	16
1.1. Introduction	17
2. The µKOS System	20
2.1. µKOS System architecture	21
2.1.1. The Application layer	21
2.1.2. The Library layer	23
2.1.3. The Manager layer (HAL)	23
2.2. The µKOS System organization	24
2.2.1. System	24
2.2.2. The Ports	25
2.3. The µKOS modules	26
2.3.1. The tool	26
2.3.2. The protocol	51
2.4. Structure of the system & application modules	54
2.4.1. Example, a system module	55
2.4.2. Example, an application module	58
2.5. Hardware requirement for supporting µKOS	71
2.6. References	71
2.7. Links	71
3. The system calls	72
3.1. The µKOS system calls	73
3.2. Rules and constraints	73

3.3. The lib_comm library	74
3.3.1. “comm” communication manager system calls	79
3.3.2. “urtx” communication manager system calls	88
3.3.3. “usbx” communication manager system calls	96
3.3.4. “bltx” communication manager system calls	103
3.4. The lib_gene library	111
3.4.1. Architecture of the non volatile memory	112
3.4.2. “iotx” manager system calls	115
3.4.3. “text” manager system calls	118
3.4.4. “syos” manager system calls	122
3.4.5. “usdc” manager system calls	137
3.4.6. “xfer” manager system calls	145
3.5. The lib_tbox library	152
3.5.1. “glob” manager system calls	155
3.5.2. “misc” manager system calls	164
3.5.3. “strg” manager system calls	173
3.6. The lib_peri library	185
3.6.1. “visx” manager system calls	188
3.6.2. “imgx” manager system calls	195
3.6.3. “adcx” manager system calls	209
3.6.4. “tmpx” manager system calls	212
3.6.5. “i2cx” manager system calls	216
3.6.6. “imux” manager system calls	223
3.6.7. “ims2” manager system calls	228
3.7. The lib_engi library	241

3.7.1. “sqee” Sysquake Embedded Engine	242
3.8. References	245
3.9. Links	245
4. The µKernel	246
4.1. Introduction	247
4.2. Basic concepts	248
4.2.1. One CPU but many processes	248
4.2.2. Temporal aspects	261
4.2.3. Reentrancy	261
4.2.4. Preemption	262
4.2.5. Inversion of priority	262
4.2.6. Critical resources	263
4.3. The µKOS µKernel implementation	264
4.3.1. State of a process	265
4.3.2. Process descriptors and lists	266
4.3.3. Daemons	269
4.3.4. Overlay processes	270
4.3.5. Interruptions	272
4.3.6. The scheduler	274
4.3.7. Semaphores	275
4.3.8. Associations - Shared Memory	279
4.3.9. Mailboxes & queues	280
4.4. Statistics and µKernel performances	282

4.4.1. Statistics	282
4.4.2. µKernel performances / measurements	283
4.5. Support of the multithread newlib	288
4.6. Customization by stubs	289
4.6.1. The kernel model for the ARM Cortex M3	290
4.6.2. The kernel model for the ARM Cortex M4	311
4.6.3. The kernel model for the ARM Cortex M7	332
4.6.4. The kernel model for the IcyCAM (icyflex-1)	353
4.7. Groups of functions of the µKernel (families of system calls)	370
4.7.1. µKernel system calls	375
4.8. References	445
4.9. Links	445
5. Event analysis	446
5.1. Event analysis	447
5.1.1. The tool uKOSLog	447
5.1.2. The µKOS macros	469
5.1.3. A complete example	470
6. Sysquake Embedded	476
6.1. A new way to compute	477
6.1.1. Using sqee with a command line	477
6.1.2. Using sqee with a C program	478
6.2. LME Tutorial	485

6.2.1. Simple operations	485
6.2.2. Complex Numbers	486
6.2.3. Vectors and Matrices	489
6.2.4. Polynomials	494
6.2.5. Strings	494
6.2.6. Variables	495
6.2.7. Loops and Conditional Execution	496
6.2.8. Functions	497
6.2.9. Local and Global Variables	499
7. The cortex-M7F	502
7.1. The cortex-M7F board	503
7.2. The CPU unit	504
7.3. Pins / interfaces configuration	506
7.4. PCB and the interfaces	519
7.4.1. The Power supply	519
7.4.2. JTAG and the (FTDI) communications	520
7.4.3. The main industrial interface	521
7.4.4. The accumulator and the Bluetooth interface	521
7.4.5. The extension bus	522
7.5. Memory mapping and CPU initialization	525

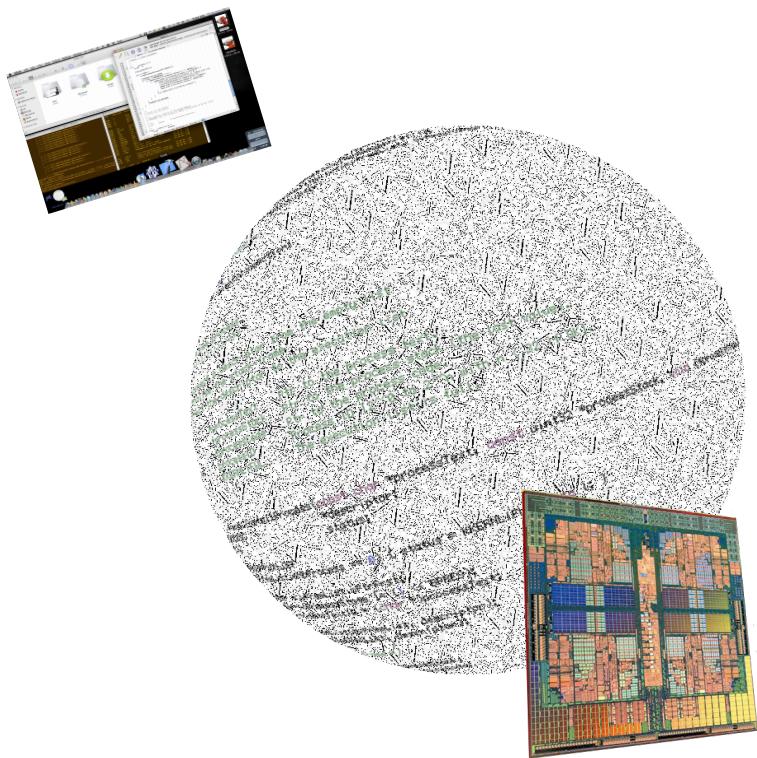
7.5.1. µKOS mapping	525
7.5.2. The CPU initialization	526
A. Annex	540
A.1. A full application	541
A.1.1. Main specifications	541
A.2. The Baphomet hardware (A)	542
A.2.1. The temperature sensor	542
A.3. The Baphomet software (B)	543
A.3.1. “tmp0” manager system calls	544
A.3.2. The background process	559
A.3.3. The protocol	564
A.4. The Sysquake software (C)	567
B. Annex	572
B.1. A standalone application	573
B.1.1. Building phase	574
B.1.2. The cmpl_std.c file	575
B.1.3. The hard.c file	588
C. Annex	592
C.1. Making ROMable applications	593
C.2. How to proceed	593

C.2.1. Programming rules	594
C.2.2. Adaptation of the application.c	595
C.2.3. Adaptation of the main µKOS makefile	596
D. Annex	598
D.1. Introduction	599
D.2. Considered target: ST Discovery 429 board	599
D.2.1. Preparing the software package	600
D.2.2. Modify the necessary files	601
D.2.3. Building the new system	602
E. Annex	604
E.1. Rules and conventions used for the project	605
E.2. The package	606
E.2.1. Prerequisites for OSX (10.12.x)	606
E.2.2. Prerequisites for Ubuntu (16.04 LTS)	609
E.2.3. Download & install the necessary open source packages	614
E.2.4. Install the “cross-compilers”	614
E.2.5. Build additional µKOS UNIX tools	615
E.2.6. Build the full µKOS system	615
E.2.7. Build the standalone µKernel application	615
E.2.8. Build all the example applications	616
E.2.9. Build the “doxygen” documentation	616
E.3. Setting-up the Eclipse environment	617

E.3.1. Download the Eclipse package	617
E.3.2. Configuring Eclipse	617
F. Annex	622
F.1. The gcc compiler	623
F.1.1. Download the necessary packages	623
F.1.2. Prepare the environment	623
F.1.3. Setting-up the compiler (t-arm-elf)	630
F.1.4. Building the packages in this order	631
F.1.5. Script for building OSX native gcc compilers	632
F.1.6. Script for building cpu32 cross-compilers	635
F.1.7. Script for building coldfire cross-compilers	639
F.1.8. Script for building blackfin cross-compilers	643
F.1.9. Script for building ARM cross-compilers (M3 - M4 - M7)	647
F.1.10. Script for building icyflex1 cross-compilers	651
F.1.11. Intermediate scripts for building the binutils, gcc, newlib and gdb	655
G. Annex	664
G.1. uKOS-III MISRA C 1998 compliance matrix	666
G.2. References	677

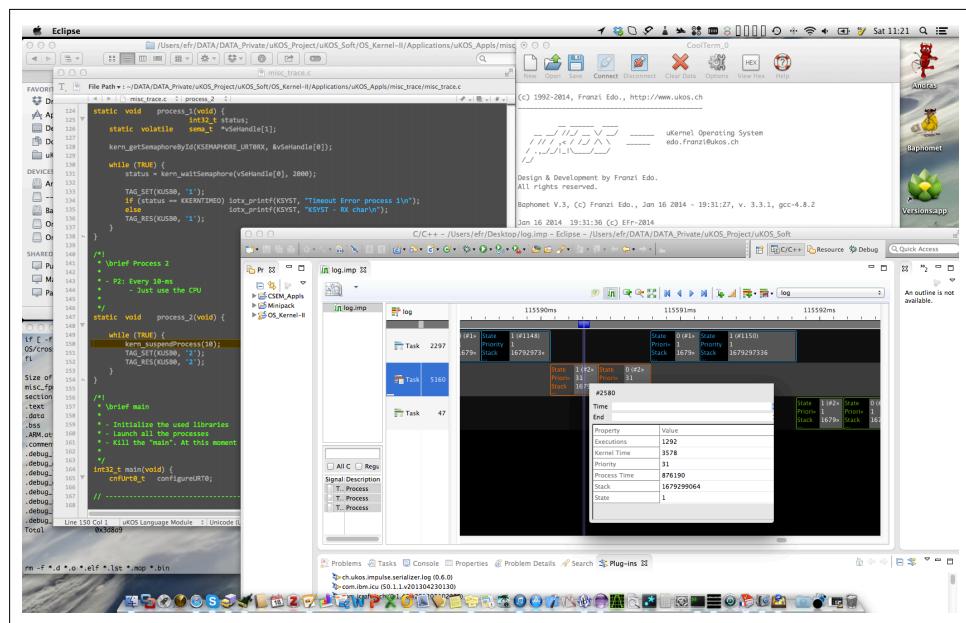
v 4.1

1. Introduction



1.1. Introduction

μKOS is a multitasking OS suitable for small embedded μController or DSP systems. I developed μKOS early in 1992 during the **Khepera** mobile robot project. When I designed Khepera I had to think about “how to write code to control it”. So, this was the starting point of μKOS. The first implementation was written in **CALM assembler** (a LAMI-EPFL tool). I completely rewrote μKOS in C in 1996; the used host computer was a Mac running OS9 with the **MPW** environment. In 2002 Apple switched to OSX (UNIX world) and I decided to reactivate, to update and to complete it, in order to share the ideas and the experience cumulated during this project.



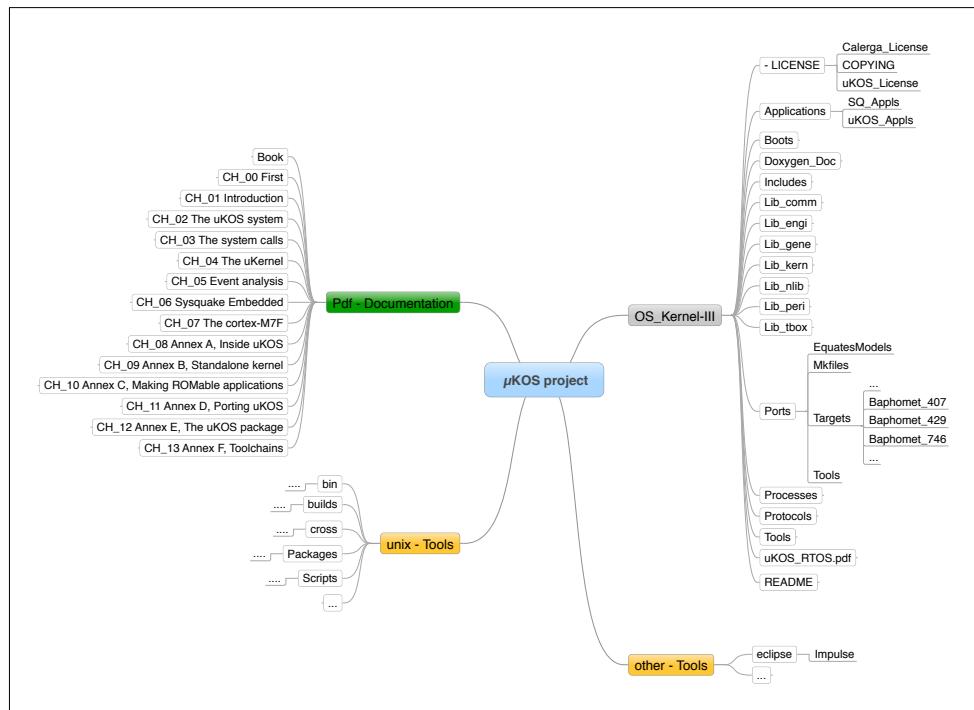
μKOS was designed with in mind the idea to put in connection a target (μController, DSP, etc.) responsible for handling the real time constraints of an application, with a workstation running complex software for scientific data manipulation such as **Matlab**, **Sysquake**, **Mathematica** or **LabView**. The connection between the target and the workstation can be as simple as a RS232 link or a more complex interface like USB or Ethernet. The original idea was to exchange between the target and the workstation only high-level messages. Initially the connection was supported by a RS232 link; this was very helpful for some debugging phases because it was possible to control the target directly via a VT100 text terminal.

I consider this project an open source (under GPL3 licence). So, all the documents (schematics, sources and the documentation) can be free downloaded from the web site <http://www.ukos.ch>.

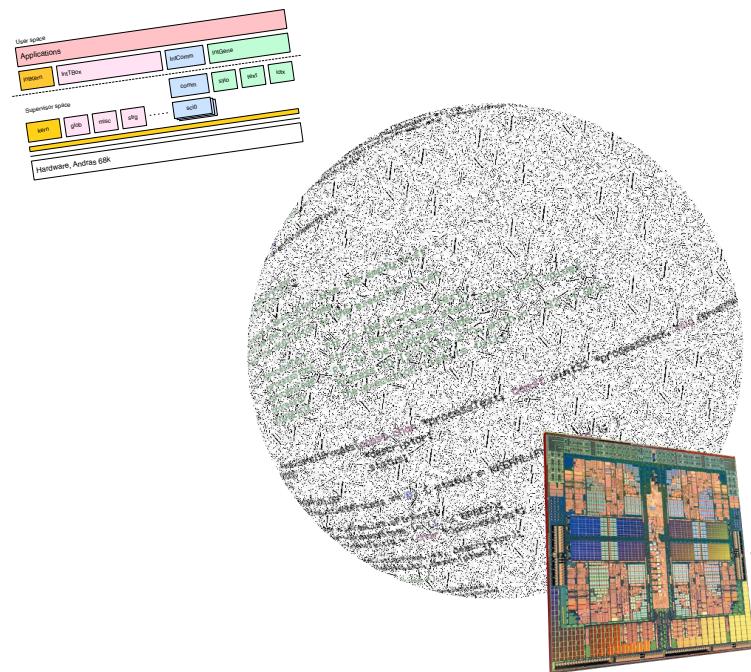
The project is described and distributed for a well-known ARM μController of the **cortex M3-M4-M7** family. The porting on the CSEM icyflex-1 by the **IcyCAM** SoC is also presented. It is obvious that the μKOS RTOS can be ported to a more modern μController or DSP.

The purpose of this book (more a project report) is to describe all the components involved in an embedded microprocessor system; the **hardware**, the **system** (OS), the **μKernel**, the **libraries** (managers) and the **gcc tools**. This document is addressed to all people having a very good knowledge in digital electronics as well as in computer science fields.

Here is the view on all the topics covered by the μKOS project.

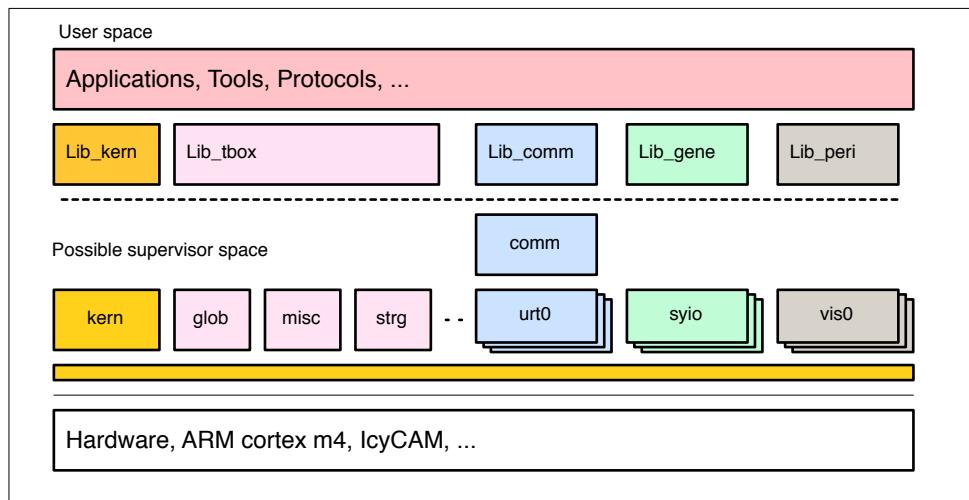


2. The μKOS System



2.1. μKOS System architecture

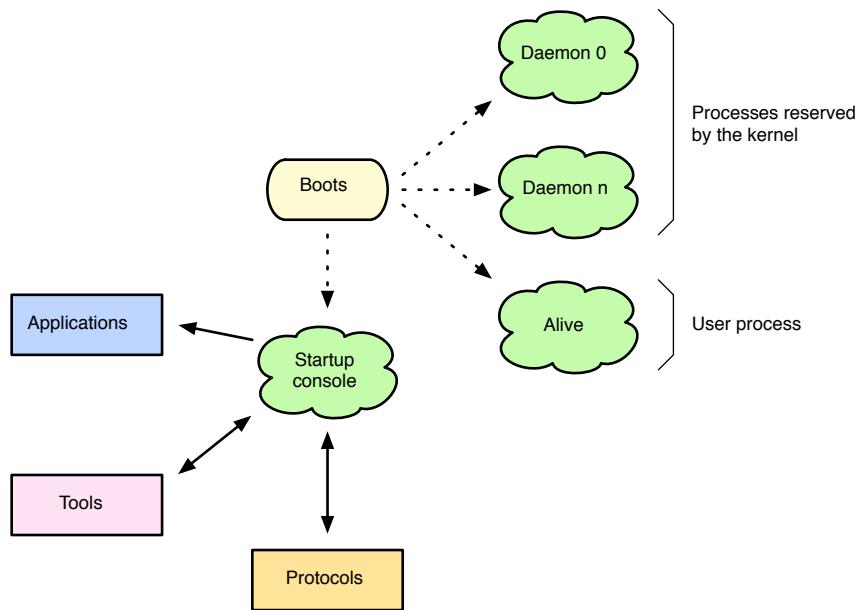
μKOS RTOS architecture is based on **hierarchical layers of modules**. At the top of the architecture there is the **Application** layer (by application is intended all the high-level modules capable to exploit the hardware and the software resources of the system). This layer is organized in families of modules. Under the Application layer there is the **Library** layer. This layer allows the interface between the **Application** layer and the different **managers** (BIOS). The **Manager** layer is the **HAL** (Hardware Abstraction Layer) implementing the interface between the system call functions and the physical world (chips, interfaces, electronics, etc.). Moreover, it should allow a minimal effort of work when the system has to be ported onto other platforms.



2.1.1. The Application layer

The Application layer is composed of families of modules designed to have a specific behavior. In the μKOS implementation four families are considered: the **Processes**, the **Overlays**, the **Protocols** and the **Tools**.

Logic of the families of modules



- Processes:** this family of modules concerns all the standalone processes (or daemons) capable to be launched and running concurrently (multi-tasking).
- Overlays:** this family of modules concerns all the standalone processes (or daemons) capable to be launched and running concurrently (multi-tasking). The overlays are processes that need to be reloaded before their execution.
- Protocols:** this family of modules is designed to permit an easy interface between a host computer running a high-level analysis software such as Matlab, Sysquake or LabView and the board running μKOS responsible for handling the real-time of the application.
- Tools:** this family of modules is composed of any kind of usable applications (i.e., the Motorola S or Intel Hex loaders, console, process, etc.).

2.1.2. The Library layer

The Library layer **Lib_xyzs** is simple the redirection of the different high-level system calls to the corresponding **managers**.

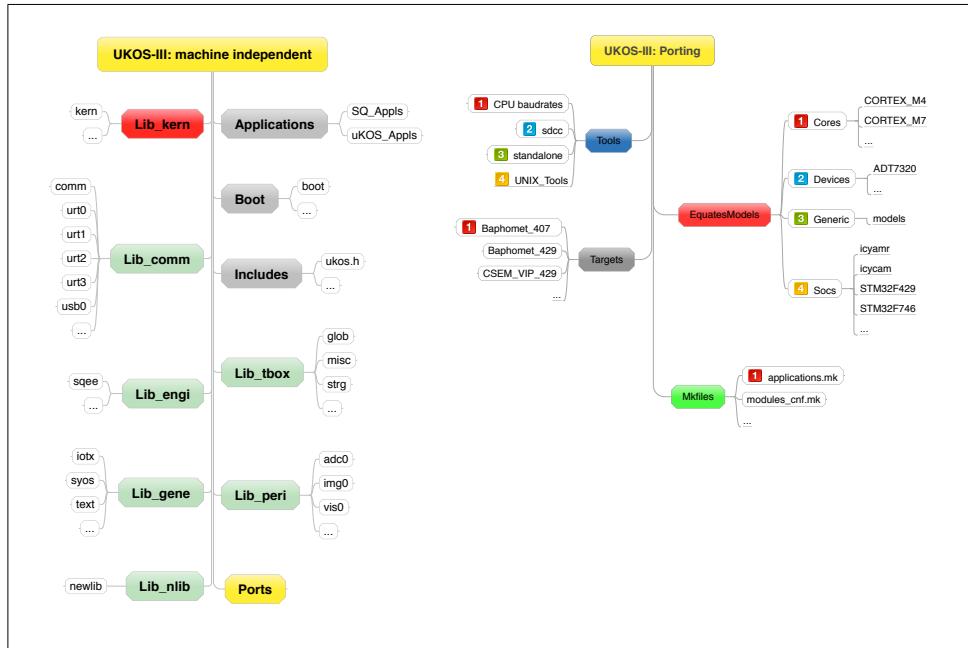
2.1.3. The Manager layer (HAL)

The Manager layer is the **HAL** (Hardware Abstraction Layer) of the system. It is composed of families of functions designed to handle the hardware of the system. The μKOS implementation knows the following families: the **kern**, the **comm**, the **gene**, the **tbox**, the **peri** and the **nlib**.

1. **kern**: this family of modules is responsible to implement the multitasking and the synchronization of all the events of the system. This module is the central pillar of the system.
2. **comm**: this family of modules is responsible to handle and to make coherent all the I/O communication managers such as the usb, the uart, the sci, etc.
3. **gene**: this family of modules is responsible to handle the generic services such the high-level text manipulation or the management of the **mms** (memory mass storage) of the system.
4. **tbox**: this family of modules is similar to the **gene** one. It is maintained separate from the **gene** family only for ensuring the μKOS backward compatibility.
5. **peri**: this family of modules is responsible to operate with the different I/O of the system. All the peripherals such as the ADC converters, the vision sensors or the imagers are under the control of this family.
6. **nlib**: this family of modules handles the interface between the generic Red-Hat newlib (configured for multi-thread operations) and the different Managers.

2.2. The μKOS System organization

2.2.1. System



The system is organized around two main trees of folders/files; the **μKOS-III** (machine independent) and the **Ports** (machine/architecture specific).

The **μKOS-III** folders/files contain all the pieces of code and definitions which are independent of an hardware or an architecture. All the system calls are prepared at this level before to be dispatched to the specific porting part for the final hardware customization.

The **Ports** folders/files contain all the pieces of code and definitions specifics to a hardware implementations or a CPU architecture. The Porting folders/files contain all the detailed functionalities of a system. Four folders contain the main pieces of the system infrastructure; the **EquatesModels**, the **Mkfiles**, the **Tools** and the **Targets**.

2.2.2. The Ports

EquatesModels

This folder contains all the equates (chip definitions, macros, etc.) and all the models (uarts, i2c, scared, etc.) of the system: more in detail:

1. **Cores:** this folder contains all the hardware independent but a core specific libraries (i.e. cortex-m4, cortex-m7, icyflex-1, etc.).
2. **Devices:** this folder contains all the peripheral chips definitions used by the project.
3. **Generic:** this folder contains all the functionality models that can be reused in many projects (i.e. the temperature and the SDCard management).
4. **SOCs:** this folders contains all the SOC specific definitions, macros, models, etc.

Targets

This folder contains all the pieces of code necessary to customize a target. In this folder we have at least a **Base** and a **Variant_1** folders. The Base folder contains the implementations of all functionalities available on the main CPU board. Variant_1, 2, ..x contain all the possible derivatives of the target (i.e. different crystal, more memory, less LEDs, etc.).

Mkfile

This folder contains all the generic makefiles.

Tools

This folder contains some tools usable for creating the systems or for the development phase: more in detail:

1. **CPU baudrates:** this folder contains an Excel sheet usable to compute the baud rates for some micro controller.
2. **sdcc:** this folder contains some 8-bit applications (i.e. for PICs).
3. **standalone:** this folder contains the s-loader engine usable for downloading and for executing the code during the development phase.
4. **UNIX_tools:** this folder contains some specific UNIX terminal tools usable by the makefiles.

2.3. The μKOS modules

2.3.1. The tool

The tool modules are useful for working with the μKOS environment. The console is responsible for collecting formatted messages coming from a communication manager (usb0, uart0, sci0, etc.), and calling the specific module of the application layer with the associated parameters. The **console** can deal with the following list of modules; this list is not exhaustive because μKOS is an evolutive and customizable system.

Tools	
bench	Execute some bench tests
binfill	Fill a memory with a raw binary format file
console	Console (multi-user)
cycle	Launch cyclically a tool
dump	Memory dump
dumptrace	Dump of the sos debug trace
echo	Echo on a communication manager
fill	Fill a memory with a pattern
hexloader	Intel hex format loader
kill	Kill a process
list	Display the list of the available modules
man	Display the help for the selected module
memory	Display the memory usage
memx	Mass Memory Storage management
mpy	Run the MicroPython embedded engine
pdp11loader	Dec PDP11 format loader
process	Display the list of the running processes
restart	Restart the OS
run	Run a function module
sloader	Motorola S format loader

Tools	
sqe	Run the Sysquake embedded engine
test	Minimal LED and memory tests
uartx	Configure the baud-rate of the urtx communication manager
uKOS	Give information about μKOS

Details of the Tool modules



bench

Execute some benches.

Usage from the **console** (this module is THREAD-SAFE):

bench

Example:

```
bench
Nov 17 2012 14:25:53 (c) EFr-2017
Bench xx: xyz
```

binfill

Download a raw binary format file via a communication manager. By definition the raw binary has any format. The module waits for 10-s a raw flow. After 10-s, if no data has been transferred, an error is signaled.

Usage from the **console** (this module is THREAD-SAFE):

```
binfill
```

Example:

```
binfill
Raw binary format loader mode.
Nov 17 2012 14:25:53 (c) EFr-2017
1234567890
BIN: download terminated checksum = 0x20D
```

console

Multi-user command line interpreter. As many as 4 simultaneous consoles can be active at the same time.

Usage from the **console** (this module is not THREAD-SAFE):

```
console {commManager, urt0, urtn, usb2 ...}
```

Example:

```
console urt1
Jun 11 2013 21:56:46 (c) EFr-2013
Console.
```

```
uKOS >
```

cycle

Launch cyclically a tool/protocol as a background process.

Usage from the **console** (this module is not THREAD-SAFE):

```
cycle prgmName time {commManager, urt0, urtn, usb2 ...}
```

Example:

```
cycle process 1000
cycle process 1000 urt0
Nov 17 2012 14:25:54 (c) EFr-2017
Process launched.
```

```
uKOS > Nov 17 2012 14:25:56 (c) EFr-2017
Daemon 0, idle process. (c) EFr-2017 - used cpu 99.84%
Daemon 1, supervise the process timeout. (c) EFr-2017 - used cpu 0.13%
Daemon 2, verify the stack integrity. (c) EFr-2017 - used cpu 0.0%
start-up process of the system. (c) EFr-2017 - used cpu 0.0%
alive process: the system is living. (c) EFr-2017 - used cpu 0.0%
cycle process. (c) EFr-2017 - used cpu 0.0%
```

dump

Dump a memory region.

Usage from the **console**:

dump startAdd endAdd

Example:

```
dump 400 430
0x00000400: 00,10,10,68,00,10,10,6C,00,10,10,70,00,10,10,74  ...h....l...p....t
0x00000410: 00,10,10,78,00,10,10,7C,00,10,10,80,00,10,10,84  ....x....|.....
0x00000420: 00,10,10,88,00,10,10,8C,00,10,10,90,00,10,10,94  .....
0x00000430: 00,10,10,98,00,10,10,9C,00,10,10,A0,00,00,00,00  .....
```

dumptrace

Dump the syos debug trace.

Usage from the **console**:

```
dumptrace
```

Example:

```
dumptrace
Jul  2 2017 11:09:34 (c) JMK-2017
Start of System trace dump:
Newest
--> Process 0: Generate the crash
--> Process 0: Load the registers
--> Process 0: Enter
--> Main: Enter
Oldest
End of System trace dump.
```

echo

Echo from a communication manager **x** on the communication manager **y**.

Usage from the **console** (this module is THREAD-SAFE):

```
echo commManagerx commManagery
```

Example:

```
echo urt0 urt1
echo usb0 urt0
echo urt0 urt0
Nov 17 2012 14:25:58 (c) EFr-2017
uuKKOSS
```

fill

Fill a memory region with a pattern.

Usage from the **console** (this module is THREAD-SAFE):

```
fill startAdd endAdd data
```

Example:

```
fill 1A34 2346 5A  
Nov 17 2012 14:25:54 (c) EFr-2017
```

hexloader

Download an Intel hex format file via a communication manager, leave the cmdLine environment and execute the downloaded application

Usage from the **console** (this module is not THREAD-SAFE):

```
hexloader {-run | -norun}
```

Example:

```
hexloader
hexloader -run
hexloader -norun
```

kill

Kill an installed process.

Usage from the **console** (this module is THREAD-SAFE):

```
kill processName
```

Example:

```
kill Process_alive
Nov 17 2012 14:25:55 (c) EFr-2017
```

list

Display the list of the available modules.

Usage from the **console** (this module is THREAD-SAFE):

```
list familyId
```

Example:

```
list
list x
Nov 17 2012 14:25:55 (c) EFr-2017
mms0 8 XBF_ v1.0 binfill Raw binary loader. (c) EFr-2017
mms0 9 XDP_ v1.0 dump Dump a memory area. (c) EFr-2017
mms0 10 XFI_ v1.0 fill Fill a memory area with a pattern. (c) EFr-2017
mms0 11 XHL_ v1.0 hexloader Intel hex+ (32-bit) loader. (c) EFr-2017
mms0 12 XKI_ v1.0 kill Kill a running process. (c) EFr-2017
mms0 13 XLS_ v1.0 list List the system modules. (c) EFr-2017
mms0 14 XMN_ v1.0 man Show the help of the module. (c) EFr-2017
mms0 15 XME_ v1.0 memory Give the memory section information. (c) EFr-2017
mms0 16 XMX_ v1.0 memx Management of the mass storage unit. (c) EFr-2017
mms0 17 XPR_ v1.0 process List the installed processes. (c) EFr-2017
mms0 18 XRZ_ v1.0 restart Make a restart. (c) EFr-2017
mms0 19 XRN_ v1.0 run Run a downloaded code. (c) EFr-2017
mms0 20 XuK_ v1.0 uKOS The uKOS information. (c) EFr-2017
mms0 21 Xu0_ v1.0 uart0 Set the baud-rate of the urt0 manager. (c) EFr-2017
mms0 22 Xu1_ v1.0 uart1 Set the baud-rate of the urt1 manager. (c) EFr-2017
mms0 23 Xu2_ v1.0 uart2 Set the baud-rate of the urt2 manager. (c) EFr-2017
mms0 24 Xu3_ v1.0 uart3 Set the baud-rate of the urt3 manager. (c) EFr-2017
mms0 26 XSL_ v1.0 sloader Motorola S1-9, S2-8, S3-7 loader. (c) EFr-2017
mms0 27 XCY_ v1.0 cycle cycle function. (c) EFr-2017
mms0 28 XEC_ v1.0 echo echo function. (c) EFr-2017
```

man

Display the help for the selected module.

Usage from the **console** (this module is THREAD-SAFE):

```
man {moduleName}
```

Example:

```
man
man process
Nov 17 2012 14:25:56 (c) EFr-2017
```

```
List the installed processes
=====
```

```
This tool displays some information concerning
the installed processes.
```

```
Input format: process {-all | -noall}
Output format: Process information
```

```
uKOS uKernel is (c) of Franzi Edo., 1992-2017
All rights reserved
edo.franzi@ukos.ch
```

memory

Display the memory usage.

Usage from the **console** (this module is THREAD-SAFE):

memory

Example:

```
memory
Nov 17 2012 14:25:56 (c) EFr-2017
Section text:    address: 0x08000188, length: 0x0000A360
Section rodata: address: 0x0800A4E8, length: 0x000044FC
Section data:   address: 0x20000000, length: 0x00000850
Section bss:    address: 0x20000958, length: 0x0001B1D0

Internal heap:  address: 0x20000000, used:  0.0%
External heap: address: 0x90000000, used:  0.0%
```

memx

Management of the Memory Mass Storage x. This command allow to program or to erase a memory block.

Usage from the **console** (this module is THREAD-SAFE):

```
memx -mem1 [-E | -W | A] index
```

Example:

```
memx -mem1 -W 0
memx -mem1 -E 2
memx -mem1 -A
memx -mem1 -A 3      It erases all the blocs from the 3 to the n
Nov 17 2012 14:25:56 (c) EFr-2017
Terminate.
```

mpy

Launch the MicroPython Process.

Usage from the **console** (this module is not THREAD-SAFE):

```
mpy urt1 -internal 90000
```

Example:

```
mpy urt2 -external 400000
mpy urt1 -internal 90000
```

```
uKOS interface for Micro-Python (www.micropython.com)
Package 1.9 for uKOS-III
```

```
> import math
> math.Pi
> 3.14
```

pdp11loader

Download a Dec pdp11 format file via a communication manager, live the cmdLine environment and execute the downloaded application

Usage from the **console** (this module is not THREAD-SAFE):

```
pdp11loader {-run | -norun}
```

Example:

```
pdp11loader
pdp11loader -run
pdp11loader -norun
```

process

Display the list of the running processes.

Usage from the **console**:

```
process {-all | -noall}
```

Example:

```
process
process -all
process -noall
Nov 17 2012 14:25:56 (c) EFr-2017
Daemon 0, idle process.          (c) EFr-2017 - used cpu 99.85%
Daemon 1, supervise the process timeout. (c) EFr-2017 - used cpu 0.13%
Daemon 2, verify the stack integrity. (c) EFr-2017 - used cpu 0.0%
start-up process of the system.      (c) EFr-2017 - used cpu 0.0%
```

restart

Restart the OS.

Usage from the **console** (this module is THREAD-SAFE):

restart

Example:

restart Nov 17 2012 14:25:57 (c) EFr-2017

(c) 1992-2017, Franz Edo., <http://www.ukos.ch>

uKernel Operating System
edo.franzi@ukos.ch

Design & Development by Franzi Edo.

All rights reserved.

Discovery 429 (Variant_Test), V.4.8.0, gcc-7.1.0, svn: 168:178M
(c) Franzi Edo., Aug 4 2017 - 11:47:33
sw = 0

Aug 4 2017 11:47:37 (c) EFr-2017
Console.

JKOS >

run

Run a downloaded code.

Usage from the **console** (this module is THREAD-SAFE):

run

Example:

run

sloader

Download a Motorola S format file via a communication manager, live the cmdLine environment and execute the downloaded application.

Usage from the **console** (this module is not THREAD-SAFE):

```
sloader {{-run | -norun}, {-sdcard sector}}
```

Example:

```
sloader
sloader -run
sloader -sdcard 2345
```

sqe

Launch the Sysquake Embedded Process.

Usage from the **console** (this module is not THREAD-SAFE):

```
sqe urt1 -internal 90000
```

Example:

```
sqe urt2 -external 400000
sqe urt1 -internal 90000
```

```
Mar  8 2016 12:00:13 (c) EFr-2017
uKOS interface for Sysquake Embedded (Calerga Sarl, www.calerga.com)
LME 7.0 (build Mar  7 2016 23:51:30 f32)
```

```
> rand(4,3)
ans =
  0.236    0.2631    0.2969
  0.7214    0.9245    0.7002
  0.4465    0.7271    0.9628
  0.2232    0.3343    0.8977
```

```
>
```

test

Minimal tests of the LEDs and the memory

Usage from the **console** (this module is not THREAD-SAFE):

```
test {nbTests}
```

Example:

```
test 5
Sep 20 2014 00:32:22 (c) EFr-2017

Test LEDs
8-bits: Test Memory: fill 0xAA - 0x55
.....
8-bits: Test Memory: fill with an incremental pattern
.....
16-bits: Test Memory: fill 0xAA55 - 0x55AA
.....
16-bits: Test Memory: fill with an incremental pattern
.....
32-bits: Test Memory: fill 0xAA55AA55 - 0x55AA55AA
.....
32-bits: Test Memory: fill with an incremental pattern
.....
Test terminated
```

uartx

Configure the baud-rate of the urtx communication manager.

Available baud-rates: 2400, 4800, 9600, 19200, 38400,
 57600, 115200, 230400, 460800,
 500000, 921600, 1000000, 1500000,
 2000000, 2500000, 3000000 [bits/s]

Usage from the **console**:

uartx baud-rate

Example:

```
uart0 115200
uart1 230400
Nov 17 2012 14:25:58 (c) EFr-2017
Change the terminal to 230400-bits/s.
```

uKOS

Display the µKOS information.

Usage from the **console** (this module is THREAD-SAFE):

uKOS

Example:

uKOS

```
Nov 17 2012 14:25:57 (c) EFr-2017
uKOS is a development environment for embedded
real-time applications.
```

```
uKOS is an I.P. of Franzzi Edo.
```

```
      5-Route de Cheseaux
      CH-1400 Cheseaux-Noréaz
```

```
email: edo.franzzi@ukos.ch
```

2.3.2. The protocol

The protocol modules are useful for connecting a target with a high level software such as Matlab, Sysquake or LabView. the protocols have a strict format which is simple to be decoded. Here are some examples.

Protocols	
W	Write data to a peripheral bus
R	Read data from a peripheral bus
L	Control the LED states
B	Give the revision of the TBOX manager and of the ROM

Details of the Protocol modules



W

Write a data to a peripheral bus.

Usage from the **console** (this module is THREAD-SAFE):

w, data, offset

Example:

w, 34, 126

w, 12, 345

w

R

Read a data from a peripheral bus.

Usage from the **console** (this module is THREAD-SAFE):

R,offset

Example:

R,126

R,345

r,12

L

Control the LED states.

Available parameter: action

0: turn off the LED

1: turn on the LED

2: toggle the LED

Usage from the **console** (this module is THREAD-SAFE):

L,ledNumber,action

Example:

L,1,0

L,1,2

1

B

Give the revision of the TBOX manager, the KERN manager and of the ROM.

Usage from the **console** (this module is THREAD-SAFE):

B

Example:

B

b,5.0.1,3.1,2.0

2.4. Structure of the system & application modules

Both, μKOS system and application modules have to declare a dedicated table (structure of data) containing information and properties of the module. Here is **module** structure:

```
struct module {
    uint32_t     oIdModule;           // Module identifier
    const char_t *oStrApplication;   // Ptr on the application string
    const char_t *oStrHelp;          // Ptr on the help string
    int32_t      (*oExecution) \     // Function pointer to the execution code
                  (uint32_t argc, \
                   char_t *argv[]);
    char_t       *oStrRevision;       // Program Revision
    uint32_t      oFlag;             // Module flag
#define        BSHOW      0           // Visible in help
#define        BEXECONSOLE 1         // Executable from the CLI
#define        BCONFIDENTIAL 2       // No comment
};

};
```

The element **oldModule** is the unique **identifier** of the module. The **MSW** represents the family **ID** and the **LSW** the **unique ID** of the module (see the file **modules.h**). The element **oStrApplication** is the pointer on the **short description string** of the module (this string is visible when we call the **list** tool). The element **oStrHelp** is the pointer on the **self-contained help** of the module. The element **oStrRevision** is the pointer on the **revision** of the module (this string is visible when we call the **man** tool). Here are some examples:

```
list
Nov 17 2012 14:25:55 (c) EFr-2017
mms0 8 XBF_ v1.0 binfill Raw binary loader. (c) EFr-2017
mms0 9 XDP_ v1.0 dump Dump a memory area. (c) EFr-2017
mms0 10 XFI_ v1.0 fill Fill a memory area with a pattern. (c) EFr-2017
mms0 11 XHL_ v1.0 hexloader Intel hex+ (32-bit) loader. (c) EFr-2017
mms0 12 XKI_ v1.0 kill Kill a running process. (c) EFr-2017
mms0 13 XLS_ v1.0 list List the system modules. (c) EFr-2017
```

The element **oExecution** is the entry pointer of the module (with the **argc** and **argv** arguments). The element **oFlags** reflects the behavior of the module. All the flags can be combined together.

Architecture of a system module:

2.4.1. Example, a system module

```
/*
; man.
; ----

;-----;
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113              $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Show the help of the module.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          / \   / \   / \   / \
;   5-Route de Cheseaux  / / / , < / / / / \_ \
;   CH 1400 Cheseaux-Noréaz / / / / | / _/ / _/ /
;                           \_,/_/ |_\_//__/
;
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include      <uKOS.h>
```

```
// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "man           Show the help of the module.          (c) EFr-2017";
LOC_CONST_STRG(aStrHelp[]) =
    "Show the help of the module\n"
    "=====\\n\\n"
    "This tool displays the help for a\\n"
    "selected module.\\n\\n"
    "Input format: man {moduleName}\\n"
    "Output format: [result]\\n\\n";

LOC_CONST_STRG(aStrLogo[]) =      KSTRLOGO_HELP;
LOC_CONST_STRG(aStrCopyright[]) = KCOPYRIGHT_FRANZI

// Prototypes
// =====

static int32_t      prgm(uint32_t argc, char_t *argv[]);

// Module specifications (See the modules.h)
// =====

MODULE(Man, KIDTOOL, KMANNUM, prgm, "1.0", ((1<<BSHOW) | (1<<BEXECONSOLE)))

/*
 * \brief Main entry point
 *
 */
static int32_t      prgm(uint32_t argc, char_t *argv[]) {
    uint16_t      index;
    uint32_t      idModule;
    module_t      *module;
    bool_t        error = FALSE;

    iotx_printf(KSYST, __DATE__ "  __TIME__ (c) EFr-2017\\n\\n");
```

```
if (argc == 2) {  
  
    // man with parameters  
  
    if (syos_getModuleName(KMEM0, (char_t *)argv[1], &index, \  
        &module) != KSYOSNOERR) {  
        if (syos_reserve() != KSYOSNOERR)  
            error = TRUE;  
        else {  
            if (syos_getModuleName(KMEM1, (char_t *)argv[1], \  
                &index, &module) != KSYOSNOERR) {  
                syos_release();  
                error = TRUE;  
            }  
        }  
    }  
    if (!error)  
        iotx_printf(KSYST, "%s%s", module->oStrHelp, aStrCopyright);  
}  
  
// man without parameter  
// Display the logo and the tool list  
  
else {  
    iotx_printf(KSYST, "%s", aStrLogo);  
  
    index = 0;  
    while (syos_getModuleFa(KMEM0, KIDTOOL, & idModule, &index, \  
        &module) == KSYOSNOERR) {  
        if (module->oFlag & (1<<BSHOW))  
            iotx_printf(KSYST, "%s\n", module->oStrApplication);  
  
        index++;  
    }  
}  
  
if (!error) { iotx_printf(KSYST, "\n");  
    return (EXIT_SUCCESS_OS); }  
  
else {  
    iotx_printf(KSYST, "Protocol error.\n\n");  
    return (EXIT_FAILURE);  
}  
}
```

2.4.2. Example, an application module

```
/*
; header.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113             $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        header for the uKOS applications.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          / \ / \ / \ / \
;   5-Route de Cheseaux  / / / , < / / / / \_ \
;   CH 1400 Cheseaux-Noréaz / / / / | / _/ / _/ /
;                           \_,/_/ |_\_//_/
;
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
    .extern      _start
    .extern      _lnApplication
    .extern      aUserAppl_Specifications

    .section     .text
```

```
/* Header */
/* ----- */

.long 0x6D656D31          /* mem1                         */
.long _start                 /* Entry point                   */
.long _lnApplication        /* Length of the application   */
.long aUserAppl_Specifications /* Module specifications      */
```

```
/*
; crt0_App.
; =====

;-----
; Author:      Franzi Edo.  The 2007-05-27
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113             $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        crt0 for the uKOS applications.
;              See "ld.pdf" file, "Using LD, the GNU linker" page 48.
;
; (c) 1992-2017, Franzi Edo.
; -----
;
; Franzi Edo.          _____/_____\_____
; 5-Route de Cheseaux   / / / ,< / / / / \_ \
; CH 1400 Cheseaux-Noréaz / / / / | / / / / \_ / /
;                         \_,/_/ |_\_\_//__/
;edo.franzi@ukos.ch

;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include     <uKOS.h>
#include     <EPROM.ck>
```

```
#if (defined(__CPLUSPLUS__))
typedef      void          (*fnct_t)(void);
typedef      fnct_t        *fnctCpp_t;

extern      fnct_t        _stCTOR, _enCTOR;
extern      fnct_t        _stdTOR, _endTOR;
#endif

extern      uint32_t      _stDATA, _enDATA, _enRODATA;
extern      uint32_t      _stBSS, _enBSS;

/*
 * \brief _start
 *
 * - Copy the initialized data from the CODE to the DATA region
 * - Initialize the BSS region
 * - Call the main
 *
 */
int32_t      _start(void) {
    uint8_t      *regionDATA, *regionCODE, *regionBSS;
    int32_t      nbBytes, status;
    uint32_t      ckSumm;

    #if (defined(__CPLUSPLUS__))
    fnctCpp_t    ctor, dtor;
    #endif

    // Copy the initialized data from the CODE region to the DATA one

    regionDATA = (uint8_t *)&_stDATA; regionCODE = (uint8_t *)&_enRODATA;
    nbBytes = (int32_t)&_enDATA - (int32_t)&_stDATA;
    while (nbBytes-- > 0) { *regionDATA++ = *regionCODE++; }

    // Initialize the BSS region

    regionBSS = (uint8_t *)&_stBSS;
    nbBytes = (int32_t)&_enBSS - (int32_t)&_stBSS;
    while (nbBytes-- > 0) { *regionBSS++ = 0; }

    // Verify if the application is compatible with the burned OS

    #if (defined(__CHECK_OS_VERSION__))
    syos_getCKSumOS(&ckSumm);
    if (ckSumm != KOSCKSUMM) {
        return (EXIT_FAILURE_CRT0);
    }
    #endif
```

```
// Call the constructors

#if (defined(__CPLUSPLUS__))
ctor = &_stCTOR;
while (ctor < &_enCTOR) { (*ctor)(); ctor++; }
#endif

// Verify if the user memory is available

RESERVE(SYOS, KDEVALL);
status = main();

// Call the destructors

#if (defined(__CPLUSPLUS__))
dtor = &_stDTOR;
while (dtor < &_enDTOR) { (*dtor)(); dtor++; }
#endif

return (status);
}
```

```
/*
; asso_smpl.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113             $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:    uKOS
; Goal:       Demo of a C application.
;             This application shows how to operate with the uKOS uKernel.
;             - P0: Every 1000-ms
;                   Toggle LED 0
;             - P1: Publish an association named "counter" with the tic-count
;                   Every 1000-ms
;                   Read the tic-count
;                   After 20 loops, P1 will commit a suicide
;             - P2: Look for an association named "counter"
;                   Every 500-ms
;                   Display the information
;
;             (c) 1992-2017, Franzi Edo.
; -----
;
; Franzi Edo.          _ _ _ / / / _ \ \ _ /
; 5-Route de Cheseaux  / / / , < / / / / \ _ \
; CH 1400 Cheseaux-Noréaz / / / / | / / / / _ / /
;                         \_,/_ / |_\_ / / _ / /
;
; edo.franzi@ukos.ch

;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
; -----
*/

```

```
#include      <uKOS.h>

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "asso_smpl    Example 1 of how to use associations.      (c) EFr-2017";

LOC_CONST_STRG(aStrHelp[]) =
    "This is a ROMable C application\n"
    "=====\\n\\n"

    "This user function module is a C written application.\\n\\n"

    "Input format:  asso_simple\\n"
    "Output format: [result]\\n\\n";

// Module specifications (See the modules.h)
// =====

MODULE(UserApp1, KIDAPPLICATION, KAPPLICATIONNUM, _start, "1.0", \
((1<<BSHOW) | (1<<BEXECONSOLE)))

/*
 * \brief Process 0
 *
 * - P0: - Every 1000-ms
 *       - Toggle LED 0
 *
 */
static void  process_0(const void *argument) {

    while (TRUE) {
        kern_suspendProcess(5000);
        misc_toggleLed(0);
    }
}
```

```
/*
 * \brief Process 1
 *
 * - P1: - Publish an association named "counter" with the tic-count
 *       - Every 1000-ms
 *       - Read the tic-count
 *       - After 20 loops, P1 will commit a suicide
 *
 */
static void process_1(const void *argument) {
    uint8_t i;
    uint64_t counter;
    shar_t pack;
    volatile proc_t *process;
    volatile astp_t *association;

    kern_getTiccount(&counter);

    pack.oGeneral = (void *)&counter;
    pack.oSize = sizeof(uint64_t);

    if (kern_createAssociation("Counter", &association) != KKERNNOERR) {
        exit(EXIT_FAILURE);
    }
    if (kern_publishAssociation(association, &pack) != KKERNNOERR) {
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < 20; i++) {
        kern_suspendProcess(1000);
        misc_toggleLed(1);
        kern_getTiccount(&counter);
    }

    // It is time to commit a suicide ...
    kern_getProcessRun(&process);
    kern_killProcess(process);
    while (TRUE);
}
```

```
/*
 * \brief Process 2
 *
 * - P2: - Look for an association named "counter"
 *       - Every 500-ms
 *       - Display the information
 *
 */
static void process_2(const void *argument) {
    uint64_t      *counter;
    shar_t        *pack;
    volatile astp_t *association;

// Waiting for the association OK

    while (kern_getAssociationById("Counter", &association) != KKERNNOERR) {
        kern_switchFast();
    }
    while (kern_findAssociation(association, &pack)           != KKERNNOERR) {
        kern_switchFast();
    }

    counter = (uint64_t *)pack->oGeneral;

// The association is effective; link a pointer on it

    while (TRUE) {
        kern_suspendProcess(500);
        iotx_printf(KSYST, "Counter = %ld\n", *counter);
    }
}
```

```
/*
 * \brief main
 *
 * - Initialize the used libraries
 * - Launch all the processes
 * - Kill the "main". At this moment only the launched processes are executed
 *
 */
int32_t      main(void) {
    volatile     proc_t          *process_0, *process_1, *process_2;

    LOC_CONST_STRG(aStrIden_0[]) =      "Process_User_0";
    LOC_CONST_STRG(aStrIden_1[]) =      "Process_User_1";
    LOC_CONST_STRG(aStrIden_2[]) =      "Process_User_2";
    LOC_CONST_STRG(aStrText_0[]) =      "User 0 process,           (c) EFr-2017";
    LOC_CONST_STRG(aStrText_1[]) =      "User 1 process,           (c) EFr-2017";
    LOC_CONST_STRG(aStrText_2[]) =      "User 2 process,           (c) EFr-2017";

// Specifications for the processes

    PROC_SUPV(0, vSpecification_0, aStrText_0, KSZSTACKMIN, process_0, \
              aStrIden_0, KDEF0, 31, 0, KPROCNORMAL);
    PROC_SUPV(1, vSpecification_1, aStrText_1, KSZSTACKMIN, process_1, \
              aStrIden_1, KDEF0, 1, 0, KPROCNORMAL);
    PROC_SUPV(2, vSpecification_2, aStrText_2, KSZSTACKMIN, process_2, \
              aStrIden_2, KDEF0, 1, 0, KPROCNORMAL);

    if (kern_createProcess(&vSpecification_0, &vProcess_0) != KKERNNOERR)
        exit(EXIT_FAILURE);
    if (kern_createProcess(&vSpecification_1, &vProcess_1) != KKERNNOERR)
        exit(EXIT_FAILURE);
    if (kern_createProcess(&vSpecification_2, &vProcess_2) != KKERNNOERR)
        exit(EXIT_FAILURE);
    return (EXIT_SUCCESS_OS);
}
```

```
/*
; link_App.
; =====

;-----
; Author:      Franz Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113              $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Mapping for the uKOS applications. It uses the external RAM.
;
;   (c) 1992-2017, Franz Edo.
; -----
;
;   Franz Edo.          _____/ / / / \ \ / / /
;   5-Route de Cheseaux  / / / / ,< / / / /\_ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / / \_ / /
;                           \_,/_ / |_\ \_ / / \_ / /
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
ENTRY(_start)

_lnApplication      = ((_enRODATA - _stTEXT) + (__exidx_end - _stDATA));

MEMORY {
    prgm_code (rx)      : ORIGIN = 0xD0200000, LENGTH = 4096K
    prgm_data (rwx)     : ORIGIN = 0xD0600000, LENGTH = 2048k
}

SECTIONS {
    .text : {
        _stTEXT = ABSOLUTE(.);
        *(.text .text.*)
        _enTEXT = ABSOLUTE(.);
        _stRODATA = ABSOLUTE(.);
        *(.rodata .rodata.*)
        . = ALIGN(4);
        _enRODATA = ABSOLUTE(.);

        _stCTOR = ABSOLUTE(.);
        KEEP(*(SORT(.init_array.*)))
        KEEP(*(.init_array))
        _enCTOR = ABSOLUTE(.);

        _stDTOR = ABSOLUTE(.);
        KEEP(*(SORT(.fini_array.*)))
        KEEP(*(.fini_array))
        _enDTOR = ABSOLUTE(.);
    } > prgm_code

    .data : AT (_enRODATA) {
        _stDATA = ABSOLUTE(.);
        *(.data .data.*)
        . = ALIGN(4);
        *(vtable)
        *(.data*)
        _enDATA = ABSOLUTE(.);
    } > prgm_data

    . = ALIGN(4);
    .eh_frame : {
        KEEP (*(.eh_frame))
    } > prgm_data

    . = ALIGN(4);
    .ARM.extab : {
        *(.ARM.extab*)
    } > prgm_data
}
```

```
. = ALIGN(4);
__exidx_start = ABSOLUTE(.);
.ARM.exidx : {
    *(.ARM.exidx*)
} > prgm_data
__exidx_end = ABSOLUTE(.);

.bss (NOLOAD) : {
    _stBSS = ABSOLUTE(.);
*(.bss .bss.*)
    . = ALIGN(4);
    *(COMMON)
    . = ALIGN(4);
    _enBSS = ABSOLUTE(.);
    _end = ABSOLUTE(.);
    end = ABSOLUTE(.);
} > prgm_data
}
```

2.5. Hardware requirement for supporting μKOS

μKOS can be ported on different hardware platforms. Here is the list of the minimal hardware requirement for supporting the OS and the μKernel:

1. 16-32-bit **CPU** having different interruption priorities. The CPU should have to be supported by the **gcc**.
2. The CPU should be provided with a software interruption (**traps**).
3. The **Flash/EEPROM/ROM...** memory should have at least 64-KBytes.
4. The **RAM** memory should have at least 64-KBytes of capacity.
5. At least these peripherals should be available: **2 LEDs, 2 timers, 2 jumpers, 1 RS232**.
6. All the peripherals should have the capability to generate interruptions. Moreover, it is **mandatory** that the **2 timers** required by the μKernel have a **lower interruption priority** in respect of the priority used by the other peripherals.

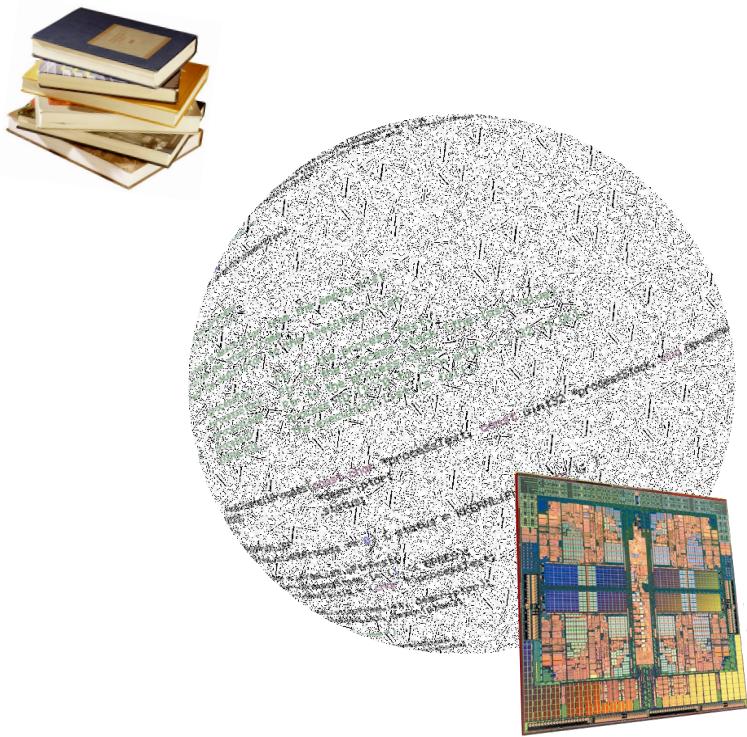
2.6. References

- Labrosse Jean J.**, "μC/OS-II, The Real-Time Kernel" Lawrence, Kansas, R&D Technical Books, 1998.
Barr Michael, "Programming Embedded Systems in C and C++" O'Reilly, 1999.
Simon David E., "An Embedded Software Primer" Addison-Wesley, 2000.
Van Sickle Ted, "Programming Microcontrollers in C, 2nd ed." LLH, 2000.

2.7. Links

- <http://www.gnu.org/>
<http://lists.gnu.org/>
<http://sourceware.org/newlib/>
<http://www.embedded.com/>
<http://www.netrino.com/Publications/>
<http://www.billgatliff.com/>

3. The system calls



3.1. The μKOS system calls

The μKOS system is built around libraries of system calls organized in managers. A library can contain one or more managers. μKOS defines six families of libraries: the **lib_kern**, **lib_comm**, **lib_gene**, **lib_tbox**, **lib_peri**, and the **lib_engi**.

The **lib_kern** library is the main core of the system. The μKernel is responsible for organizing the multitasking and the CPU time resource control. This library is presented in a separate document (see the chapter 4).

The **lib_comm** library is responsible for handling (and eventually redirecting) all the communication to the available hardware supported by the manager (i.e. “**scio**” for the SCI Serial Communication Interface, “**urt0**”, “**urt1**” for the Universal asynchronous Receiver Transmitter, etc.). This allows to introduce an abstraction layer and an easy and immediate switch between all the system communications to the selected manager.

The **lib_gene** library is responsible for handling high-level system calls that facilitate the work with the system.

The **lib_tbox** library is responsible for handling the generic low level resources of the board (LEDs, switches, serial EEPROM, etc.).

The **lib_peri** library is responsible for handling all the access to the available I/O peripherals (i.e. “**img0**” for the imager peripheral, “**adc0**” for the analog to digital peripheral, “**vis0**” for the vision sensor peripheral, etc.).

The **lib_engi** library is responsible for handling third part computing engines like **spice**, **Octave** or **Sysquake**.

3.2. Rules and constraints

Before entering into the details of all the libraries and managers, it is necessary to introduce the basic rules used for the implementation of the system call mechanism.

```
Prototype:    int32_t xyzt_function(type param1, type param2, ...)
Call:         status = xyzt_function(param1, param2, ...);
```

1. **xyzt** represents the name of the manager
2. All the functions return a **int32_t** status. **NULL** returns indicate no error; negative returns indicate some special situations (usually errors).

3.3. The lib_comm library

This library controls all the communication I/O of the system. The user can directly use a specific manager like “**sci0**”, “**urt0**”, “**usb0**”, etc.. The generic “**comm**” manager allows to redirect the calls to the appropriate communication manager according to the value of the **commManager** parameter. Here are some examples:

1. At the **construction of the system** the communication manager by default is the **usb3**
2. At the **creation of the process** the communication manager is the **urt8**

```
KURTO comm_xyz(KURTO, ...);      redirect the call to      urt0_xyz(...);
KSCI3  comm_xyz(KSCI3, ...);      redirect the call to      sci3_xyz(...);
KDEF0  comm_xyz(KDEF0, ...);      redirect the call to      usb3_xyz(...);
KSYST  comm_xyz(KSYST, ...);     redirect the call to      urt8_xyz(...);
```

All the communication managers have the same set of system calls with the same I/O parameters. The configuration is done via the manager specific structures. Here is an example:

```
// Generic configuration structure

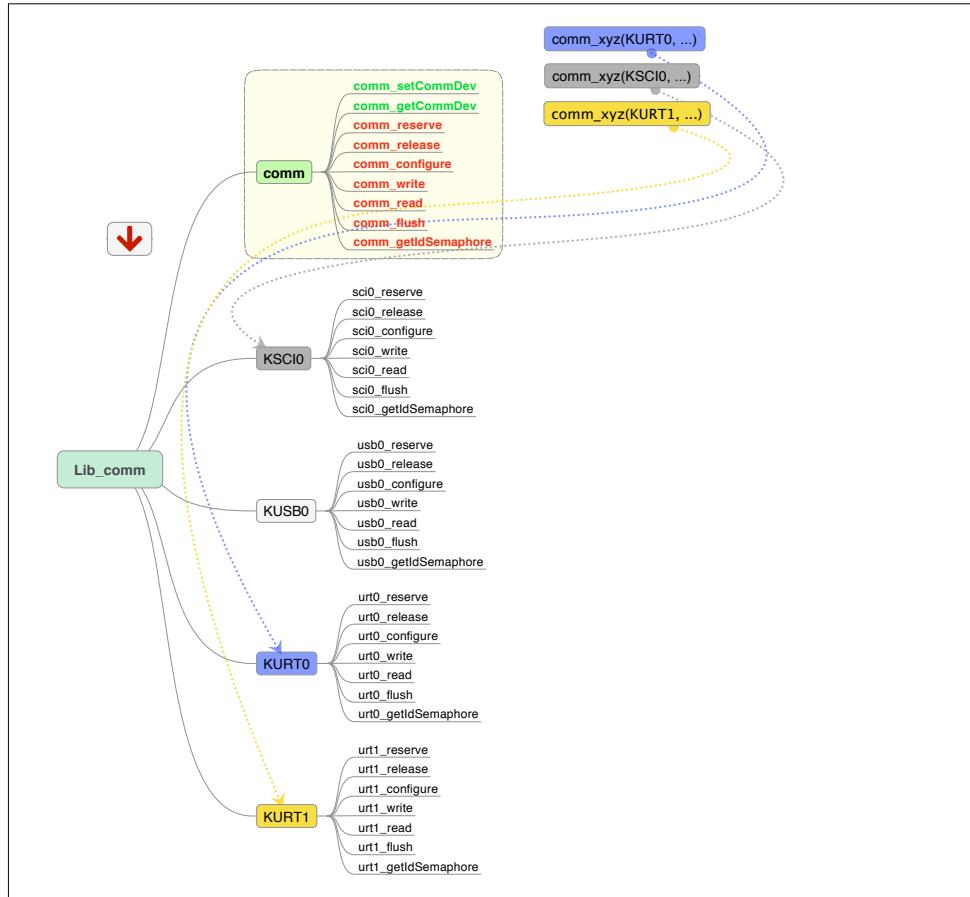
const cnfUrtx_t    configure;

configuration.oBaudRate = KBD115200;
configuration.oKernSync = (1<<BRXSEMA);
configuration.oNBits    = K8BIT;
configuration.oStopBits = K1STBIT;
configuration.oParity   = KNONE;
...

This call ...
comm_configure(KURTO, (void *)&configure);

... is equivalent to this one.

urt0_configure(&configure);
```



comm System Calls	
comm_reserve	Reserve the communication manager
comm_release	Release the communication manager
comm_configure	Configure the communication manager
comm_write	Write a buffer to the communication manager
comm_read	Read a buffer from the communication manager
comm_flush	Flush the communication manager
comm_getIdSemaphore	Get the RX-TX semaphore id of the manager
comm_getCommDev	Get the communication manager
comm_setCommDev	Set the communication manager

urtx System Calls	
urtx_reserve	Reserve the urtx communication manager
urtx_release	Release the urtx communication manager
urtx_configure	Configure the urtx communication manager
urtx_write	Write a buffer to the urtx communication manager
urtx_read	Read a buffer on the urtx communication manager
urtx_flush	Flush of the urtx communication manager
urtx_getIdSemaphore	Get the RX-TX semaphore id of the urtx communication manager

usbx System Calls	
usbx_reserve	Reserve the usbx communication manager
usbx_release	Release the usbx communication manager
usbx_configure	Configure the usbx communication manager
usbx_write	Write a buffer to the usbx communication manager
usbx_read	Read a buffer on the usbx communication manager
usbx_flush	Flush of the usbx communication manager
usbx_getIdSemaphore	Get the RX-TX semaphore id of the usbx communication manager

bltx System Calls	
bltx_reserve	Reserve the bltx communication manager
bltx_release	Release the bltx communication manager
bltx_configure	Configure the bltx communication manager
bltx_write	Write a buffer to the bltx communication manager
bltx_read	Read a buffer on the bltx communication manager
bltx_flush	Flush of the bltx communication manager
bltx_getIdSemaphore	Get the RX-TX semaphore id of the bltx communication manager

To facilitate the control of the communication managers, a couple of macros are available to ensure the coherence of the information flow during the context switching of the μKernel (**thread safe**). Here is an example: 2 processes use the same output communication manager to communicate.

```
#define      RESERVE_COMM(commManager, mode) \
            while (comm_reserve(commManager, mode) == KCOMMCHBSY) kern_switchFast();

#define      RELEASE_COMM(commManager, mode) \
            comm_release(commManager, mode);

/*
 * Process 0
 *
 * - Every 1000-ms
 *   Reserve the communication manager by default (KDEF0) in write mode
 *   Print a string
 */
void process_0(const void *argument) {

    while (TRUE) {
        kern_suspendProcess(1000);
        RESERVE_COMM(KDEF0, KDEVWRITE);
        comm_write(KDEF0, "String of the process 0\n", 24);
        RELEASE_COMM(KDEF0, KDEVWRITE);
    }
}

/*
 * Process 1
 *
 * - Every 879-ms
 *   Reserve the communication manager specified
 *   during the process creation (KSYST) in write mode
 *   Print a string
 */
void process_1(const void *argument) {

    while (TRUE) {
        kern_suspendProcess(879);
        RESERVE_COMM(KSYST, KDEVWRITE);
        comm_write(KSYST, "String of the process 1\n", 24);
        RELEASE_COMM(KSYST, KDEVWRITE);
    }
}
```

3.3.1. “comm” communication manager system calls

```
int32_t comm_reserve(uint32_t commManager,
                      uint8_t mode)
```



Reserve the communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
	mode	KDEVREAD	Only for read operations
		KDEVWRITE	Only for write operations
		KDEVALL	For read and write operations
output:	status	KCOMMXXXXX	Depends on the "xxxx" communication manager
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
int32_t      status;
...
status = comm_reserve(KDEF0, KCOMMWRITE);
...
while (comm_reserve(KDEF0, KDEVWRITE) == KCOMMCHBSY) {
    kern_switchFast();
}
comm_xyz();
status = comm_release(KDEF0, KDEVWRITE);
```

This can be rewritten:

```
RESERVE_COMM(KDEF0, KDEVWRITE);
comm_xyz();
RELEASE_COMM(KDEF0, KDEVWRITE);
```

```
int32_t comm_release(uint32_t commManager,
                      uint8_t mode)
```

Release the communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
	mode	KDEVREAD	Only for read operations
		KDEVWRITE	Only for write operations
		KDEVALL	For read and write operations
output:	status	KCOMMXXXXX	Depends on the "xxxx" communication manager
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
int32_t      status;
...
status = comm_release(KDEF0, KDEVWRITE);
```

```
int32_t comm_configure(uint32_t commManager,
                      const void *configure)
```

Configure the communication manager.

input:	commManager	KDEF0 KSYST KURTO KXYZn	Communication manager (default) Communication manager set at the process creation Communication manager (i.e. force the usage of the urt0 communication manager) Communication manager XYZn
	*configure		Ptr on the configuration buffer
output:	status	KCOMMXXXXX	Depends on the "xxxx" communication manager
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
int32_t      status;
const cnfUrtx_t  configure = {
    .oBaudRate = KBD57600,
    .oKernSync = (1<<BRXSEMA),
    .oNBits   = K8BIT,
    .oStopBits = K1STBIT,
    .oParity   = KNONE
};

...
status = comm_configure(KURTO, (void *)&configure);
```

```
int32_t comm_write(uint32_t commManager,
                   const uint8_t *buffer,
                   uint32_t size)
```

Write a buffer to the communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
	*buffer		Ptr on the buffer
	size		Size of the buffer
output:	status	KCOMMXXXX	Depends on the "xxxx" communication manager
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
#define KSIZE          10

uint8_t      buffer[KSIZE] = { 0,1,2,3,4,5,6,7,8,9 };
int32_t      status;
...
status = comm_write(KDEF0, buffer, KSIZE);
```

```
int32_t comm_read(uint32_t commManager,
                  uint8_t *buffer,
                  uint32_t *size)
```

Read a buffer from the communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
	*buffer		Ptr on the buffer
	*size		Ptr on the size
output:	status	KCOMMXXXX	Depends on the "xxxx" communication manager
	*size		Contains the effective number of bytes read
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
uint8_t      buffer[1];
uint32_t     size;
int32_t      status;
...
size = 1;
status = comm_read(KDEF0, buffer, &size);
```

int32_t comm_flush(uint32_t commManager)

Flush the communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
output:	status	KCOMMXXXX	Depends on the "xxxx" communication manager
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = comm_flush(KDEF0);
```

```
int32_t comm_getIdSemaphore(uint32_t commManager,
                            uint8_t semaphore,
                            char_t **identifier)
```

Get the semaphore identifier of the selected RX-TX communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
	semaphore	KSEMARMX	RX semaphore
		KSEMATX	TX semaphore
	**identifier		Ptr on the semaphore Id
output:	status	KCOMMXXXX	Depends on the "xxxx" communication manager
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
int32_t      status;
char_t       *identifier[2];
...
status = comm_getIdSemaphore(KURTO, KSEMARMX, &identifier[0]);
status = comm_getIdSemaphore(KURTO, KSEMATX, &identifier[1]);
iotx_printf(KSYST, "Semaphore ids: %s, ...%s\n", identifier[0], identifier[1]);
```

```
int32_t comm_getCommDev(uint32_t *commManager)
```

Get the communication manager.

input:	*commManager	Ptr on the communication manager
output:	status	KCOMMNOERR
reentrancy:	NO	OK
		Use RESERVE_COMM/RELEASE_COMM macros
		to be thread-safe

Call example in C:

```
uint32_t commManager;  
...  
status = comm_getCommDev(&commManager);
```

int32_t comm_setCommDev(uint32_t commManager)

Set the communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set at the process creation
		KURTO	Communication manager (i.e. force the usage of the urt0 communication manager)
		KXYZn	Communication manager XYZn
output:	status	KCOMMNOERR	OK
reentrancy:	NO		Use RESERVE_COMM/RELEASE_COMM macros to be thread-safe

Call example in C:

```
...
status = comm_getCommDev(KURTO);
```

3.3.2. “urt~~x~~” communication manager system calls



int32_t urtx_reserve(uint8_t mode)

Reserve the **urt~~x~~** manager.

input:	mode	KDEVREAD	Only for read operations
		KDEVWRITE	Only for write operations
		KDEVALL	For read and write operations
output:	status	KURTxNOERR	The manager is reserved
		KURTxCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
...
status = urtx_reserve(KDEVWRITE);
...
while (urtx_reserve(KDEVWRITE) == KURTxCHBSY) {
    kern_switchFast();
}
urtx_xyz();
status = urtx_release(KDEVWRITE);
```

This can be rewritten:

```
RESERVE(URTx, KDEVWRITE);
urtx_xyz();
RELEASE(URTx, KDEVWRITE);
```

```
int32_t urtx_release(uint8_t mode)
```

Release the **urtx** manager.

input:	mode	KDEVREAD	Only for read operations
		KDEVWRITE	Only for write operations
		KDEVALL	For read and write operations
output:	status	KURT x NOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = urtx_release(KDEVWRITE);
```

```
int32_t urtx_configure(const cnfUrtx_t *configure)
```

Configure the **urtx** manager. The semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

" urtx - RX char "	Reception of a byte	
" urtx - TX char "	A message buffer was sent	
input:	*configure	Ptr on the configuration buffer
output:	status	KURTxNOERR OK
		KURTxNOCNF The configuration does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct cnfUrtx {
    uint8_t      oNBits;           // Number of bits
    #define      K7BIT      0        // K7BIT = word of 7 bits
    #define      K8BIT      1        // K8BIT = word of 8 bits

    uint8_t      oStopBits;        // Number of stop bits
    #define      K1STBIT   0        // K1STBIT = 1 stop bits
    #define      K2STBIT   1        // K2STBIT = 2 stop bits

    uint8_t      oParity;          // Parity
    #define      KNONE     0        // KEVEN = Parity even
    #define      KEVEN     1        // KODD = Parity odd
    #define      KODD      2        // KNONE = No Parity

    uint8_t      oBaudRate;        // Baudrate
    #define      KBDDEFAULT 0        // KBDDEFAULT = default baudrate
    #define      KBD1200    1        // KBD1200 = 1200 bits/s
    #define      KBD2400    2        // KBD2400 = 2400-bits/s
    #define      KBD4800    3        // KBD4800 = 4800 bits/s
    #define      KBD9600    4        // KBD9600 = 9600 bits/s
    #define      KBD19200   5        // KBD19200 = 19200 bits/s
    #define      KBD38400   6        // KBD38400 = 38400 bits/s
    #define      KBD57600   7        // KBD57600 = 57600 bits/s
    #define      KBD115200  8        // KBD115200 = 115200 bits/s
    #define      KBD230400  9        // KBD230400 = 230400 bits/s
```

```
// Optional baud-rates: supported only by some specific hardware
// If not supported, the baud-rate is not changed (previous configuration)

#define      KBD460800    10    // KBD460800 = 460800-bits/s
#define      KBD500000    11    // KBD500000 = 500000-bits/s
#define      KBD921600    12    // KBD921600 = 921600-bits/s
#define      KBD1000000   13    // KBD1000000 = 1000000-bits/s
#define      KBD1500000   14    // KBD1500000 = 1500000-bits/s
#define      KBD1843200   15    // KBD1843200 = 1843200-bits/s
#define      KBD2000000   16    // KBD2000000 = 2000000-bits/s
#define      KBD2500000   17    // KBD2500000 = 2500000-bits/s
#define      KBD3000000   18    // KBD3000000 = 3000000-bits/s

uint8_t      oKernSync;           // Kernel synchro
#define      BRXSEMA      0      // RX semaphore
#define      BTXSEMA      1      // TX semaphore
};


```

Call example in C:

```
int32_t      status;
const cnfUrtx_t  configure = {
    .oBaudRate = KBD57600,
    .oKernSync = (1<<BRXSEMA),
    .oNBits   = K8BIT,
    .oStopBits = K1STBIT,
    .oParity  = KNONE
};
...
status = urtx_configure(&configure);
```

```
int32_t urtx_write(const uint8_t *buffer,  
                    uint32_t size)
```

Write a buffer to the **urtx** manager.

input:	*buffer	Ptr on the buffer
	size	Size of the buffer
output:	status	KURT x NOERR OK
		KURT x SEPRO The sender is busy
		KURT x LNBUB The buffer length is too big
		KURT x LNBUBO The buffer length is = 0
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define KSIZE      10  
  
uint8_t      buffer[KSIZE] = { 0,1,2,3,4,5,6,7,8,9 };  
int32_t      status;  
...  
status = urtx_write(buffer, KSIZE);
```

```
int32_t urtx_read(uint8_t *buffer,  
                  uint32_t *size)
```

Read a buffer from the **urtx** manager.

input:	* buffer	Ptr on the buffer
	* size	Ptr on the size
output:	status	OK
	KURT XNOERR	The receiver buffer is empty
	KURT XRBUEM	The receiver buffer is full
	KURT XEROVR	Overrun error
	KURT XERNOI	Noise error
	KURT XERFRA	Framing error
	KURT XERPAR	Parity error
	* size	Contains the effective number of bytes read
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
uint8_t      buffer[1];  
uint32_t     size;  
int32_t      status;  
  
...  
size = 1;  
status = urtx_read(buffer, &size);
```

`int32_t urtx_flush(void)`

Flush the `urtx` manager.

input:	-	
output:	<code>status</code>	<code>KURTxNOERR</code> OK
reentrancy:	<code>NO</code>	Use <code>RESERVE/RELEASE</code> macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = urtx_flush();
```

```
int32_t urtx_getIdSemaphore(uint8_t semaphore,
                            char_t **identifier)
```

Get the semaphore identifier of the selected RX-TX **urtx** manager.

input:	semaphore	KSEMARX	Select the RX semaphore
		KSEMATX	Select the TX semaphore
	**identifier		Ptr on the semaphore Id
output:	status	KURT X NOERR	OK
		KURT X SENOE	The semaphore does not exist
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
char_t       *identifier[2];
...
status = urtx_getIdSemaphore(KSEMARX, &identifier[0]);
status = urtx_getIdSemaphore(KSEMATX, &identifier[1]);
iotx_printf(KSYST, "Semaphore ids: %s, ...%s\n", identifier[0], identifier[1]);
```

3.3.3. “usbx” communication manager system calls



```
int32_t usbx_reserve(uint8_t mode)
```

Reserve the **usbx** manager.

input:	mode	KDEVREAD	Only for read operations
		KDEVWRITE	Only for write operations
		KDEVALL	For read and write operations
output:	status	KUSBXNOERR	The manager is reserved
		KUSBXCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
...
status = usbx_reserve(KDEVWRITE);
...
while (usbx_reserve(KDEVWRITE) == KUSBXCHBSY) {
    kern_switchFast();
}
usbx_xyz();
status = usbx_release(KDEVWRITE);
```

This can be rewritten:

```
RESERVE(USBX, KDEVWRITE);
usbx_xyz();
RELEASE(USBX, KDEVWRITE);
```

int32_t usbx_release(uint8_t mode)

Release the **usbx** manager.

input:	mode	KDEVREAD	Only for read operations
		KDEVWRITE	Only for write operations
		KDEVALL	For read and write operations
output:	status	KUSBXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = usbx_release(KDEVWRITE);
```

```
int32_t usbx_configure(const cnfUsbx_t *configure)
```

Configure the **usbx** manager. The semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

```
"usbx - RX char"      Reception of a byte  
"usbx - TX char"      A message buffer was sent
```

Caution: the semaphore “**usbx - RX char**” is not usable with this implementation of the manager.

input:	*configure	Ptr on the configuration buffer
output:	status	KUSBXNOERR OK
		KUSBXNOCNF The configuration does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct cnfUsbx {  
    uint8_t      oKernSync;          // Kernel synchro  
    #define      BRXSEMA      0        // RX semaphore  
    #define      BTXSEMA      1        // TX semaphore  
};
```

Call example in C:

```
int32_t      status;  
const cnfUsbx_t configure = {  
    .oKernSync = (1<<BRXSEMA) | (1<<BTXSEMA)  
};  
  
...  
status = usbx_configure(&configure);
```

```
int32_t usbx_write(const uint8_t *buffer,  
                    uint32_t size)
```

Write a buffer to the **usbx** manager.

input:	*buffer	Ptr on the buffer
	size	Size of the buffer
output:	status	KUSBXNOERR OK
		KUSBXSEPRO The sender is busy
		KUSBXLNBUB The buffer length is too big
		KUSBXLNBUO The buffer length is = 0
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define KSIZE          10  
  
uint8_t      buffer[KSIZE] = { 0,1,2,3,4,5,6,7,8,9 };  
int32_t      status;  
...  
status = usbx_write(buffer, KSIZE);
```

```
int32_t usbx_read(uint8_t *buffer,  
                  uint32_t *size)
```

Read a buffer from the **usbx** manager.

input:	* buffer	Ptr on the buffer
	* size	Ptr on the size
output:	status	KUSBXNOERR OK
		KUSBXRBUEM The receiver buffer is empty
		KUSBXRBFUL The receiver buffer is full
	* size	Contains the effective number of bytes read
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
uint8_t      buffer[1];  
uint32_t     size;  
int32_t      status;  
...  
size = 1;  
status = usbx_read(buffer, &size);
```

`int32_t usbx_flush(void)`

Flush the **usbx** manager.

input:	-	
output:	status	KUSBXNOERR OK
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = usbx_flush();
```

```
int32_t usbx_getIdSemaphore(uint8_t semaphore,
                            char_t **identifier)
```

Get the semaphore identifier of the selected RX-TX **usbx** manager.

input:	semaphore	KSEMARX	Select the RX semaphore
		KSEMATX	Select the TX semaphore
	**identifier		Ptr on the semaphore Id
output:	status	KUSBXNOERR	OK
		KUSBXSENOE	The semaphore does not exist
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
char_t       *identifier[2];
...
status = usbx_getIdSemaphore(KSEMARX, &identifier[0]);
status = usbx_getIdSemaphore(KSEMATX, &identifier[1]);
iotx_printf(KSYST, "Semaphore ids: %s, ...%s\n", identifier[0], identifier[1]);
```

3.3.4. “bltx” communication manager system calls



```
int32_t bltx_reserve(uint8_t mode)
```

Reserve the **bltx** manager.

input:	mode	KCOMREAD	Only for read operations
		KCOMWRITE	Only for write operations
		KCOMALL	For read and write operations
output:	status	KBLTXNOERR	The manager is reserved
		KBLTXCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = bltx_reserve(KDEVWRITE);  
...  
while (bltx_reserve(KDEVWRITE) == KBLTXCHBSY) {  
    kern_switchFast();  
}  
bltx_xyz();  
status = bltx_release(KDEVWRITE);
```

This can be rewritten:

```
RESERVE(BLTX, KDEVWRITE);  
bltx_xyz();  
RELEASE(BLTX, KDEVWRITE);
```

```
int32_t bltx_release(uint8_t mode)
```

Release the **bltx** manager.

input:	mode	KCOMREAD	Only for read operations
		KCOMWRITE	Only for write operations
		KCOMALL	For read and write operations
output:	status	KBLTXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = bltx_release(KDEVWRITER);
```

```
int32_t bltx_configure(const cnfUrtx_t *configure)
```

Configure the **bltx** manager. The semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

" bltx - RX char "	Reception of a byte	
" bltx - TX char "	A message buffer was sent	
input:	*configure	Ptr on the configuration buffer
output:	status	KBLTxNOERR OK
		KBLTxNOCNF The configuration does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct cnfUrtx {
    uint8_t      oNBits;           // Number of bits
    #define      K7BIT      0        // K7BIT = word of 7 bits
    #define      K8BIT      1        // K8BIT = word of 8 bits

    uint8_t      oStopBits;        // Number of stop bits
    #define      K1STBIT   0        // K1STBIT = 1 stop bits
    #define      K2STBIT   1        // K2STBIT = 2 stop bits

    uint8_t      oParity;          // Parity
    #define      KNONE     0        // KEVEN = Parity even
    #define      KEVEN     1        // KODD = Parity odd
    #define      KODD      2        // KNONE = No Parity

    uint8_t      oBaudRate;        // Baudrate
    #define      KBDDEFAULT 0        // KBDDEFAULT = default baudrate
    #define      KBD1200    1        // KBD1200 = 1200 bits/s
    #define      KBD2400    2        // KBD2400 = 2400-bits/s
    #define      KBD4800    3        // KBD4800 = 4800 bits/s
    #define      KBD9600    4        // KBD9600 = 9600 bits/s
    #define      KBD19200   5        // KBD19200 = 19200 bits/s
    #define      KBD38400   6        // KBD38400 = 38400 bits/s
    #define      KBD57600   7        // KBD57600 = 57600 bits/s
    #define      KBD115200  8        // KBD115200 = 115200 bits/s
    #define      KBD230400  9        // KBD230400 = 230400 bits/s
```

```
// Optional baud-rates: supported only by some specific hardware
// If not supported, the baud-rate is not changed (previous configuration)

#define      KBD460800    10    // KBD460800 = 460800-bits/s
#define      KBD500000    11    // KBD500000 = 500000-bits/s
#define      KBD921600    12    // KBD921600 = 921600-bits/s
#define      KBD1000000   13    // KBD1000000 = 1000000-bits/s
#define      KBD1500000   14    // KBD1500000 = 1500000-bits/s
#define      KBD1843200   15    // KBD1843200 = 1843200-bits/s
#define      KBD2000000   16    // KBD2000000 = 2000000-bits/s
#define      KBD2500000   17    // KBD2500000 = 2500000-bits/s
#define      KBD3000000   18    // KBD3000000 = 3000000-bits/s

uint8_t      oKernSync;           // Kernel synchro
#define      BRXSEMA      0      // RX semaphore
#define      BTXSEMA      1      // TX semaphore
};
```

Call example in C:

```
int32_t      status;
const cnfUrtx_t  configure = {
    .oBaudRate = KBD115200,
    .oKernSync = (1<<BRXSEMA),
    .oNBits   = K8BIT,
    .oStopBits = K1STBIT,
    .oParity  = KNONE
};

...
status = bltx_configure(&configure);
```

```
int32_t bltx_write(const uint8_t *buffer,  
                   uint32_t size)
```

Write a buffer to the **bltx** manager.

input:	*buffer	Ptr on the buffer
	size	Size of the buffer
output:	status	KBLTXNOERR OK
		KBLTXSEPRO The sender is busy
		KBLTXLNBUB The buffer length is too big
		KBLTXLNBUO The buffer length is = 0
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define KSIZE      10  
  
uint8_t      buffer[KSIZE] = { 0,1,2,3,4,5,6,7,8,9 };  
int32_t      status;  
...  
status = bltx_write(buffer, KSIZE);
```

```
int32_t bltx_read(uint8_t *buffer,  
                  uint32_t *size)
```

Read a buffer from the **bltx** manager.

input:	* buffer	Ptr on the buffer
	* size	Ptr on the size
output:	status	OK
	KBLT X NOERR	The receiver buffer is empty
	KBLT X RBFUL	The receiver buffer is full
	KBLT X EROVR	Overrun error
	KBLT X ERNOI	Noise error
	KBLT X ERFRA	Framing error
	KBLT X ERPAR	Parity error
	* size	Contains the effective number of bytes read
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
uint8_t      buffer[1];  
uint32_t     size;  
int32_t      status;  
  
...  
size = 1;  
status = bltx_read(buffer, &size);
```

```
int32_t bltx_flush(void)
```

Flush the **bltx** manager.

input:	-	
output:	status	KBLT X NOERR
reentrancy:	NO	OK

Use **RESERVE/RELEASE** macros
to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = bltx_flush();
```

```
int32_t bltx_getIdSemaphore(uint8_t semaphore,
                            char_t **identifier)
```

Get the semaphore identifier of the selected RX-TX **bltx** manager.

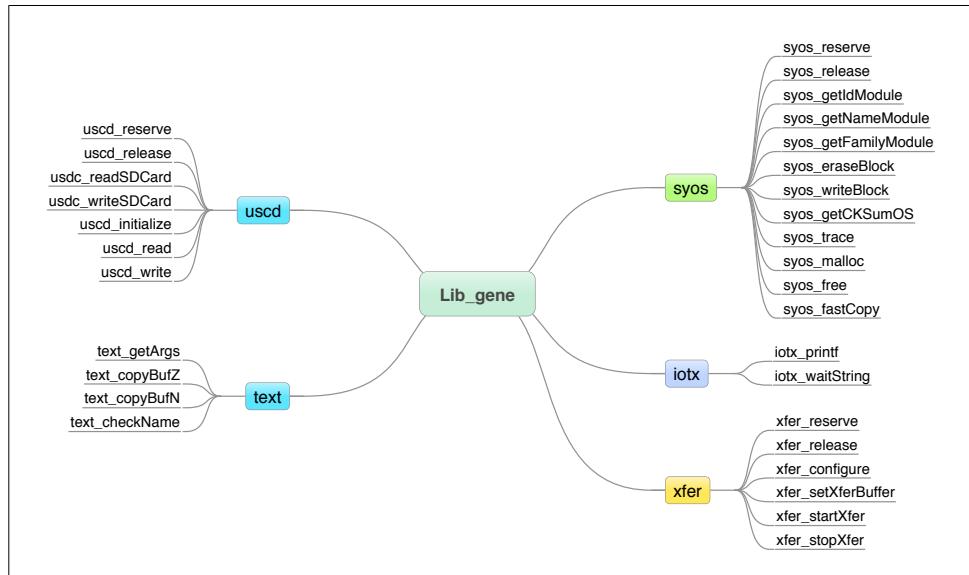
input:	semaphore	KSEMARX	Select the RX semaphore
		KSEMATX	Select the TX semaphore
	**identifier		Ptr on the semaphore Id
output:	status	KBLTXNOERR	OK
		KBLTXSENOE	The semaphore does not exist
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
char_t       *identifier[2];
...
status = bltx_getIdSemaphore(KSEMARX, &identifier[0]);
status = bltx_getIdSemaphore(KSEMATX, &identifier[1]);
iotx_printf(KSYST, "Semaphore ids: %s, ...%s\n", identifier[0], identifier[1]);
```

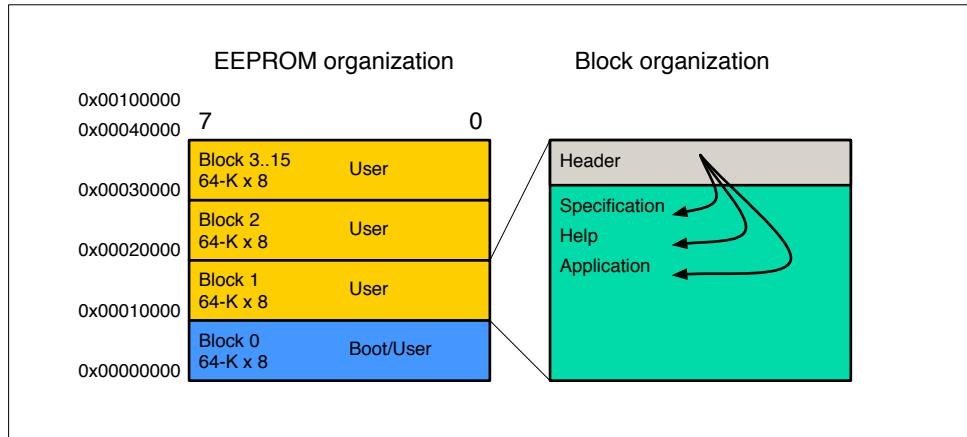
3.4. The lib_gene library

This library contains four managers; the **text**, the **iotx**, the **syos**, the **usdc** and the **xfer** one. The **text** is responsible for some text buffer manipulations such as buffer copy, or buffer compare. The **iotx** is responsible for handling some high level I/O functionalities such as the buffer printing on the standard output communication manager or waiting for an order coming from the standard input communication manager. The **syos** provides some high level functionalities of the system such as the module finding inside the concatenated list of modules of the μKOS system available inside the memory. The **usdc** is responsible to manage the Micro SDCard interface (V2).



3.4.1. Architecture of the non volatile memory

The organization of the memory is hardware (device) dependent. A group of two system calls was designed to accommodate a common access mechanism to all the memories (Flashes or EEPROMs).



```
// Header
.long      0x6D656D31          /* mem1 */
.long      _start              /* Entry point */
.long      _lnApplication     /* Length of the application */
.long      aUserAppl_Specifications /* Module specifications */

// Module specifications & help

const module_t      aUserAppl_Specifications = {
    ((KIDAPPLICATION<<24) | (KAPPLICATIONNUM<<8) | '_'),
    aStrApplication,
    aStrHelp,
    (int32_t (*)(uint32_t argc, char_t *argv[]))_start,
    "1.0",
    ((1<<BSHOW) | (1<<BEXECONSOLE))
};

LOC_CONST_STRG(aStrApplication[]) = "mbox_bidm Example of how to use ... bla bla.";
LOC_CONST_STRG(aStrHelp[]) =      "This is a ROMable C application ... bla bla\n"

// Application

int main() { ... }
```

iotx System Calls	
iotx_waitString	Waiting for a string from a communication manager
iotx_printf	Small printf
text System Calls	
text_getArgs	Get the arguments from a command line buffer
text_copyAsciiBufferZ	Copy the 2 ASCII buffers (\0 is copied)
text_copyAsciiBufferN	Copy the 2 ASCII buffers (\0 is not copied)
text_checkAsciiBuffer	Check if 2 ASCII buffers are identical
syos System Calls	
syos_reserve	Reserve the syos manager
syos_release	Release the syos manager
syos_getIdModule	Get the module header by its Id
syos_getNameModule	Get the module header by its name
syos_getFamilyModule	Get the module header by its family
syos_eraseBlock	Erase a memory block from a mass storage
syos_writeBlock	Write a memory block from to a mass storage
syos_setDoLoCode	Set the downloaded code address
syos_getDoLoCode	Get the downloaded code address
syos_getCKSumOS	Get the checksum of the OS
syos_trace	Record the trace
syos_malloc	Allocate a memory block
syos_free	Release a memory block
syos_fastCopy	Fast copy of memory blocs

usdc System Calls	
usdc_reserve	Reserve the usdc manager
usdc_release	Release the usdc manager
usdc_readSDCard	Read the SDCard
usdc_writeSDCard	Write the SDCard
usdc_initialize	Prepare the SDCard for the read/write operations
usdc_read	Read sectors
usdc_write	Write sectors

xfer System Calls	
xfer_reserve	Reserve the xfer manager
xfer_release	Release the xfer manager
xfer_configure	Configure the xfer manager
xfer_setXferBuffer	Set the transfer buffer
xfer_startXfer	Start the transfer
xfer_stopXfer	Stop the transfer

3.4.2. “iotx” manager system calls

```
int32_t iotx_waitString(uint32_t commManager,
                        char_t *ascii,
                        uint32_t size)
```



Waiting for a string (terminated by CR, LF or CR+LF) from a communication manager.

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set
		KURTO	at the process creation
		KXYZn	Communication manager (i.e. force the usage of the urt0 communication manager)
	*ascii		Communication manager XYZn
	size		Ptr on the ASCII buffer
output:	status	KIOTXNOERR	Size of the buffer
reentrancy:	THREAD-SAFE		OK

Call example in C:

```
#define KSIZE          256

int32_t      status;
char_t       ascii[KSIZE];
...
status = iotx_waitString(KDEF0, ascii, KSIZE);
```

```
int32_t iotx_printf(uint32_t commManager,
                     const char_t *format, ...)
```

Small printf. This system call does not support floating point nor 64-bit integer parameters.

Supported formats:

```
%ld, %s, %c, %x, %u

char_t *ptr = "Hello world!";
char_t *np = 0;
int32_t i = 5;
int64_t j = 5LL;
uint32_t bs  = sizeof(int32_t)*8;
uint64_t lbs = sizeof(int64_t)*8;
int32_t mi;
int64_t lmi;
char_t buf[80];

mi  = (1  << (bs-1)) + 1;
lmi = (1LL << (lbs-1)) + 1LL;

iotx_printf(KSYST, "following examples", xxx);

"%s\n", ptr                                ... Hello world!
"printf test\n"                            ... printf test
"%s is null pointer\n", np                  ... (null) is null pointer
"%d = 5\n", i                                ... 5 = 5
"%ld = 5\n", j                                ... 5 = 5
"%d = - max int\n", mi                      ... -2147483647 = - max int32
"%ld = - max int\n", lmi                    ... -9223372036854775807 = - max int64
"char %c = 'a'\n", 'a'                        ... char a = 'a'
"hex %x = ff\n", 0xff                         ... hex ff = ff
"hex %02x = 00\n", 0                           ... hex 00 = 00
"signed %d = unsigned %u = hex %x\n", \
    -3, -3, -3                                 ... signed -3 = \
                                                unsigned 4294967293 = \
                                                hex ffffffd

"signed %ld = unsigned %lu = hex %lx\n", \
    -3LL, -3LL, -3LL                           ... signed -3 = \
                                                unsigned 18446744073709551613 = \
                                                hex ffffffffffffffd
"%d %s(s)%", 0, "message"                   ... 0 message(s)
"\n"                                         ...
"%d %s(s) with %%\n", 0, "message"           ... 0 message(s) with %
```

```
"ju: \%-10s\n", "l"           ... justif: "l      "
"ju: \%10s\n", "r"           ... justif: "      r"
" 3: %04d z pad\n", 3       ... 3: 0003 z pad
" 3: %-4d l ju.\n", 3       ... 3: 3    l ju.
" 3: %4d r ju.\n", 3       ... 3:     3 r ju.
"-3: %04d z pad\n", -3      ... -3: -003 z pad
"-3: %-4d l ju.\n", -3      ... -3: -3    l ju.
"-3: %4d r ju.\n", -3      ... -3:     -3 r ju.
```

input:	commManager	KDEF0	Communication manager (default)
		KSYST	Communication manager set
		KURTO	at the process creation
		KXYZn	Communication manager (i.e. force the usage of the urt0 communication manager)
	*format		Communication manager XYZn
output:	number		Ptr on the formatted string
reentrancy:	THREAD-SAFE		Number of printed characters

Call example in C:

```
int32_t      number;
...
number = iotx_printf(KSYST, "the lazy fox ...\n");
```

3.4.3. “text” manager system calls



```
int32_t text_getArgs(char_t *ascii,
                      uint32_t size,
                      char_t *args[],
                      uint32_t *nbArgs)
```

Get the arguments from a command line.

input:	*ascii	Ptr on the ASCII buffer
	size	Size of the buffer
	*args[]	Ptr on the ascii argument buffer
	*nbArgs	Ptr on the number of ascii arguments
output:	status	KTEXTNOERR
reentrancy:	YES	OK

Call example in C:

```
const char_t      commandLine[KLNINPBUF] = "Line: 3224.5 test 123";
char_t           *argv[KLNINPBUF];
uint32_t          argc;
int32_t           status;
...
status = text_getArgs(commandLine, KLNINPBUF, argv, &argc);

iotx_printf(KSYST, "%d\n", argc);           ...    4
iotx_printf(KSYST, "%s\n", argv[0]);        ...    Line:
iotx_printf(KSYST, "%s\n", argv[1]);        ...    4.5
iotx_printf(KSYST, "%s\n", argv[2]);        ...    test
iotx_printf(KSYST, "%s\n", argv[3]);        ...    123
```

```
int32_t text_copyAsciiBufferZ(char_t *asciid,
                           const char_t *asciis)
```

Copy 2 buffers (\0 is copied).

input:	*asciid	Ptr on the ASCII destination buffer
	*asciis	Ptr on the ASCII source buffer
output:	status	KTEXTNOERR OK
reentrancy:	YES	

Call example in C:

```
#define KSIZE      256

const char_t      asciiS[] = "This is the buffer 2";
char_t          asciid[KSIZE];
int32_t        status;
...
status = text_copyBufferZ(asciid, asciiS);
```

```
int32_t text_copyAsciiBufferN(char_t *asciid,
                           const char_t *asciis)
```

Copy 2 buffers (\0 is not copied).

input:	*asciid	Ptr on the ASCII destination buffer
	*asciis	Ptr on the ASCII source buffer
output:	status	KTEXTNOERR OK
reentrancy:	YES	

Call example in C:

```
#define KSIZE      256

const char_t      asciiS[] = "This is the buffer 2";
char_t           asciid[KSIZE];
int32_t          status;

...
status = text_copyBufferN(asciid, asciiS);
```

```
int32_t text_checkAsciiBuffer(const char_t *ascii1,
                             const char_t *ascii2,
                             bool_t *answer)
```

Check if 2 ASCII buffers are identical.

input:	*ascii1	Ptr on the ASCII buffer 1
	*ascii2	Ptr on the ASCII buffer 2
	*equals	Ptr on the answer.
		The 2 ASCII buffers are identical (TRUE) or not (FALSE)
output:	status	KTEXTNOERR OK
reentrancy:	YES	

Call example in C:

```
const char_t      ascii1[] = "This is the buffer 1";
const char_t      ascii2[] = "This is the buffer 2";
int32_t          status;
bool_t           *equals;
...
status = text_checkAsciiBuffer(ascii1, ascii2, equals);
```

3.4.4. “syos” manager system calls

```
int32_t syos_reserve(uint8_t mode)
```



Reserve the **syos** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KSYOSNOERR	The manager is reserved
		KSYOSCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = syos_reserve(KDEVALL);  
...  
while (syos_reserve(KDEVALL) == KSYOSCHBSY) {  
    kern_switchFast();  
}  
syos_xyz();  
status = syos_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(SYOS, KDEVALL);  
syos_xyz();  
RELEASE(SYOS, KDEVALL);
```

```
int32_t syos_release(uint8_t mode)
```

Release the **syos** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KSYOSNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = syos_release(KDEVALL);
```

```
int32_t syos_getIdModule(uint32_t mms,
                         uint32_t idModule,
                         uint16_t *index,
                         module_t **module)
```

Get the module header by its identifier (the file modules.h contains the list of all the identifier modules).

input:	mms	The Memory Mass Storage [0..n]
	idModule	The module Id
	*index	Ptr on the filer index
	**module	Ptr on the module
output:	status	OK
	KSYOSNOERR	The memory unit does not exist
	KSYOSNOMEM	The module does not exist
	KSYOSNOMOD	Use RESERVE/RELEASE macros to be thread-safe
reentrancy:	NO	

Call example in C:

```
#define KIDMODULE ((KIDPROCESS<<16) | KSTARTUPNUM)

int32_t      status;
module_t     *module;
uint16_t     index;
...
status = syos_getIdModule(KMEMO, KIDMODULE, &index, &module);
if (status == KSYOSNOERR) {
    module->oExecution(0, 0);
}
```

```
int32_t syos_getNameModule(uint32_t mms,
                           const char_t *name,
                           uint16_t *index,
                           module_t **module)
```

Get the module header by its name.

input:	mms	The Memory Mass Storage [0..n]
	*name	Ptr on the module name
	*index	Ptr on the filer index
	**module	Ptr on the module
output:	status	KSYOSNOERR
		OK
		KSYOSNOMEM
		The memory unit does not exist
		KSYOSNOMOD
		The module does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
module_t     *module;
uint16_t     index;
...
status = syos_getNameModule(KMEMO, "sloader", &index, &module);
if (status == KSYOSNOERR) {
    module->oExecution(0, 0);
}
```

```
int32_t syos_getFamilyModule(uint32_t mms,
                            uint8_t family,
                            uint32_t *idModule,
                            uint16_t *index,
                            module_t **module)
```

Get the module header by its family and index.

input:	mms	The Memory Mass Storage [0..n]
	family	The Module family
	*idModule	Ptr on the module Id
	*index	Ptr on the filer index
	**module	Ptr on the module
output:	status	KSYOSNOERR
		OK
		KSYOSNOMEM
		The memory unit does not exist
		KSYOSNOFAM
		The family does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define KIDPRC          KIDPROCESS

int32_t      status;
uint32_t      idModule;
module_t      *module;
uint16_t      index;
...
// Install all the processes (but not the ALIVE)

index = 0;
while (syos_getFamilyModule(KMEMO, KIDPRC, &idModule, &index, &module) \
== KSYOSNOERR) {
    if (idModule != KIDALILE) {
        module->oExecution(0, 0);
    }
    index++;
}
```

```
int32_t syos_eraseBlock(uint32_t mms,
                        const uint16_t *index)
```

Erase a Memory Mass Storage block.

input:	mms *index	The Memory Mass Storage [0..n] Ptr on the filer index
output:	status	KSYOSNOERR OK KSYOSNOMEM The memory unit does not exist KSYOSNOBLK The memory block does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
uint16_t     index;
...
index = 3;
syos_eraseBlock(KMEMO, &index);
```

```
int32_t syos_writeBlock(uint32_t mms,  
                      const uint16_t *index)
```

Write a Memory Mass Storage block. The content of the memory execution area is copied into the selected Memory Mass Storage block.

input:	mms	The Memory Mass Storage [0..n]	
	*index	Ptr on the filer index	
output:	status	KSYOSNOERR OK KSYOSNOMEM KSYOSNOBLK	The memory unit does not exist The memory block does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe	

Call example in C:

```
int32_t      status;  
uint16_t     index;  
...  
index = 3;  
syos_writeBlock(KMEMO, &index);
```

```
int32_t syos_setDoLoCode(void *address)
```

Set the downloaded code address.

input:	*address	Ptr on the downloaded code
output:	status	KSYOSNOERR
reentrancy:	YES	OK

Call example in C:

```
void    *address;  
...  
address = 0x12345678;  
syos_setDoLoCode(address);
```

int32_t syos_getDoLoCode(void **address)

Get the downloaded code address.

input:	**address		Ptr on the downloaded code
output:	status	KSYOSNOERR	OK
reentrancy:	YES		

Call example in C:

```
void *address;  
...  
syos_getDoLoCode(&address);
```

```
int32_t syos_getCKSumOS(uint32_t *checksum)
```

Get the checksum of the OS.

input:	*checksum	Ptr on the checksum
output:	status	KSYOSNOERR OK
reentrancy:	YES	

Call example in C:

```
uint32_t checksum;  
...  
syos_getCKSumOS(&checksum);
```

int32_t syos_trace(const char_t *message)

Record the evolution trace of a program. This is particularly helpful for debugging purposes.

input:	*message	Ptr on the message
output:	status	KSYOSNOERR OK
output:		KSYOSSYCNA System call not available
reentrancy:	YES	

Call example in C:

```
...
syos_trace("Program call the routine _xyz\n", 0x55);
_xyz();
...
syos_trace("Program write\n", 0x24);
*myLocation = 0x00;           // i.e. this write generates a core dump
```

On the terminal we will have:

```
System dead! Core DUMP!!
=====
```

```
HardFaultException
Process_test
```

```
Program call the routine _xyz
Program write
```

```
CPU registers
r00      = 0x01234567      r08      = 0x01234567
r01      = 0x01234567      r09      = 0x01234567
r02      = 0x01234567      r10      = 0x01234567
r03      = 0x01234567      r11      = 0x01234567
r04      = 0x01234567      r12      = 0x01234567
r05      = 0x01234567      r13 (SP) = 0x01234567
r06      = 0x01234567      r14 (LR)  = 0x01234567
r07      = 0x01234567      r15 (PC)  = 0x01234567
xPSR    = 0x01234567
BASEPRI = 0x01234567
```

FPU registers

s00	= 0x01234567	s16	= 0x01234567
s01	= 0x01234567	s17	= 0x01234567
s02	= 0x01234567	s18	= 0x01234567
s03	= 0x01234567	s19	= 0x01234567
s04	= 0x01234567	s20	= 0x01234567
s05	= 0x01234567	s21	= 0x01234567
s06	= 0x01234567	s22	= 0x01234567
s07	= 0x01234567	s23	= 0x01234567
s08	= 0x01234567	s24	= 0x01234567
s09	= 0x01234567	s25	= 0x01234567
s10	= 0x01234567	s26	= 0x01234567
s11	= 0x01234567	s27	= 0x01234567
s12	= 0x01234567	s28	= 0x01234567
s13	= 0x01234567	s29	= 0x01234567
s14	= 0x01234567	s30	= 0x01234567
s15	= 0x01234567	s31	= 0x01234567
FPSCR	= 0x01234567		

```
void *syos_malloc(uint8_t location,
                  uint32_t size)
```

Memory allocation.

input:	location	KINTERNAL	Use the internal memory
		KEXTERNAL	Use the external memory
		KCCOUPLED	Use the core coupled memory
	size		Size of the memory to allocate
output:	address	!= 0	Memory address of the allocated block
	address	== 0	Memory block not allocate
reentrancy:	THREAD-SAFE		

Call example in C:

```
uint8_t      *memory;
...
memory = (uint8_t *)syos_malloc(KINTERNAL, 1234);
```

void syos_free(void *address)

Free the memory allocation.

input:	*address	Ptr on the address
output:	-	
reentrancy:	THREAD-SAFE	

Call example in C:

```
uint8_t      *memory;  
...  
syos_free(memory);
```

```
void syos_fastCopy(const void *source,
                    void *destination,
                    uint32_t nbElements,
                    uint8_t mode)
```

Fast copy of memory blocs.

input:	*source	Ptr on the source
	*destination	Ptr on the destination
	nbElements	Number of elements of 8, 16 or 32-bits
	mode	Transfer mode: 8, 16 or 32-bits
	K8BITS	8-bits
	K16BITS	16-bits
	K32BITS	32-bits
output:	status	KSYOSNOERR
		OK
		KSYOSSYNCNA
reentrancy:	NO	System call not available
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
const uint8_t      src8[123] = { 0, 1, ...34, 56};
      uint8_t      dst8[123];
const uint16_t     src16[123] = { 0, 1, ...34, 56};
      uint16_t     dst16[123];
const uint32_t     src32[123] = { 0, 1, ...34, 56};
      uint32_t     dst32[123];

...
syos_fastCopy(src8, dst8, 123, K8BITS);
syos_fastCopy(src16, dst16, 123, K16BITS);
syos_fastCopy(src32, dst32, 123, K32BITS);
```

3.4.5. “usdc” manager system calls

```
int32_t usdc_reserve(uint8_t mode)
```



Reserve the **usdc** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KUSDCNOERR	The manager is reserved
		KUSDCCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = usdc_reserve(KDEVALL);  
...  
while (usdc_reserve(KDEVALL) == KUSDCCHBSY) {  
    kern_switchFast();  
}  
usdc_xyz();  
status = usdc_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(USDC, KDEVALL);  
usdc_xyz();  
RELEASE(USDC, KDEVALL);
```

`int32_t usdc_release(uint8_t mode)`

Release the **usdc** manager.

input:	<code>mode</code>	<code>KDEVALL</code>	For read and write operations
output:	<code>status</code>	<code>KUSDCNOERR</code>	OK
reentrancy:	<code>NO</code>		Use <code>RESERVE/RELEASE</code> macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = usdc_release(KDEVALL);
```

```
int32_t usdc_readSDCard(uint8_t *buffer,  
                        uint32_t nbBytes,  
                        uint32_t sector)
```

Read data from the **SDCard**.

input:	* buffer	Ptr on the read buffer
	nbBytes	Number of bytes to read
	sector	Start sector
output:	status	KUSDCNOERR OK
		KUSDCNOCRD No SDCard
		KUSDCCANRE SDCard not recognized
		KUSDCLNBUO The number of sectors is = 0
		KUSDCTRANT Transfer not terminated
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define      KNBBYTES      54321  
int32_t    status;  
int8_t     readBuffer[KNBBYTES];  
...  
status = usdc_readSDCard(&readBuffer[], KNBBYTES, 5);  
if (status != KUSDCNOERR) {  
    exit(EXIT_FAILURE);  
}  
...
```

```
int32_t usdc_writeSDCard(const uint8_t *buffer,
                         uint32_t nbBytes,
                         uint32_t sector)
```

Write data to the **SDCard**.

input:	* buffer	Ptr on the write buffer
	nbBytes	Number of bytes to write
	sector	Start sector
output:	status	KUSDCNOERR OK
		KUSDCNOCRD No SDCard
		KUSDCCANRE SDCard not recognized
		KUSDCLNBUO The number of sectors is = 0
		KUSDCTRANT Transfer not terminated
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define      KNBBYTES      54321
        int32_t      status;
const  int8_t      writeBuffer[KNBBYTES] = { ... };
...
status = usdc_writeSDCard(&writeBuffer[], KNBBYTES, 5);
if (status != KUSDCNOERR) {
    exit(EXIT_FAILURE);
}
```

int32_t usdc_initialize(prmUsdc_t *specification)

Prepare the **usdc** manager to operate. Before using the read/write system calls it is mandatory to call this function.

input:	*specification	Ptr on the SDCard specification
output:	status	KUSDCNOERR OK
		KUSDCNOCRD No SDCard
		KUSDCCANRE SDCard not recognized
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct cnfUscd {
    uint64_t      oC_SIZE;           // Device size
    uint32_t      oPSN;              // Product Serial Number
    uint16_t      oVoltage;          // Card Voltage
    uint8_t       oMID;               // Manufacturer ID
    uint8_t      oPNM[5+1];          // Product Name
    uint8_t      oCCS;               // Card Capacity Status
#define      KCT_MMC     0x01 // MMC card Ver 3 (not supported)
#define      KCT_SD1     0x02 // SDC card Ver 1 (not supported)
#define      KCT_SD2     0x04 // SDC card Ver 2
#define      KCT_BLOCK   0x08 // Block addressing
};
```

Call example in C:

```
int32_t      status;
prmUsdc_t    sdCard;
...
status = usdc_initialize(&sdCard);
if (status != KUSDCNOERR) {
    exit(EXIT_FAILURE);
}

if (sdCard.oCCS == KCT_SD1) card = "SDCard V.1";
else                         card = "SDCard V.2";

iotx_printf(KSYST, "SDCard type           %s\n",   card);
iotx_printf(KSYST, "Manufacturer ID       %02X\n", sdCard.oMID);
iotx_printf(KSYST, "Product Name          %s\n",   sdCard.oPNM);
iotx_printf(KSYST, "Product Serial Number %04X\n", sdCard.oPSN);
iotx_printf(KSYST, "Device size [Bytes]   %ld\n",  sdCard.oC_SIZE);
iotx_printf(KSYST, "SDCard voltage         %04X\n", sdCard.oVoltage);

if (sdCard.oVoltage & 0x001) iotx_printf(KSYST, " - Supported  2.7 to 2.8V\n");
if (sdCard.oVoltage & 0x002) iotx_printf(KSYST, " - Supported  2.8 to 2.9V\n");
if (sdCard.oVoltage & 0x004) iotx_printf(KSYST, " - Supported  2.9 to 3.0V\n");
if (sdCard.oVoltage & 0x008) iotx_printf(KSYST, " - Supported  3.0 to 3.1V\n");
if (sdCard.oVoltage & 0x010) iotx_printf(KSYST, " - Supported  3.1 to 3.2V\n");
if (sdCard.oVoltage & 0x020) iotx_printf(KSYST, " - Supported  3.2 to 3.3V\n");
if (sdCard.oVoltage & 0x040) iotx_printf(KSYST, " - Supported  3.3 to 3.4V\n");
if (sdCard.oVoltage & 0x080) iotx_printf(KSYST, " - Supported  3.4 to 3.5V\n");
if (sdCard.oVoltage & 0x100) iotx_printf(KSYST, " - Supported  3.5 to 3.6V\n\n");

...
```

```
int32_t usdc_read(const prmUsdc_t *specification,
                   uint8_t *buffer,
                   uint32_t sector,
                   uint8_t nbSectors)
```

Read 1 or more sectors of 512-bytes from the SDCard. The maximum number of sector readable per call is 255.

input:	*specification	Ptr on the SDCard specification
	*buffer	Ptr on the buffer
	sector	Start sector
	nbSectors	Number of sector to read (1..255)
output:	status	KUSDCNOERR OK
		KUSDCNOCRD No SDCard
		KUSDCLNBUO The number of sectors is = 0
		KUSDCTRANT Transfer not terminated
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define      KNBSECTORS      5;

      int32_t      status;
      prmUsdc_t    sdCard;
static uint8_t      buffer[KNBSECTORS*KSZFATSECT];
...
status = usdc_initialize(&sdCard);
...
status = usdc_read(&sdCard, buffer, 45678, KNBSECTORS);
```

```
int32_t usdc_write(const prmUsdc_t *specification,
                    const uint8_t *buffer,
                    uint32_t sector,
                    uint8_t nbSectors)
```

Write 1 or more sectors of 512-bytes to the SDCard. The maximum number of sector writable by one call is 255.

input:	*specification	Ptr on the SDCard specification
	*buffer	Ptr on the buffer
	sector	Start sector
	nbSectors	Number of sector to write (1..255)
output:	status	KUSDCNOERR
		OK
		KUSDCNOCRD
		No SDCard
		KUSDCLNBUO
		The number of sectors is = 0
		KUSDCTRANT
		Transfer not terminated
reentrancy:	NO	Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
#define      KNBSECTORS     2;

        int32_t      status;
        prmUsdc_t    sdCard;
const  uint8_t     buffer[KNBSECTORS*KSZFATSECT] = { 0, 1, ... };
...
status = usdc_initialize(&sdCard);
...
status = usdc_write(&sdCard, buffer, 35, KNBSECTORS);
```

3.4.6. “xfer” manager system calls



```
int32_t xfer_reserve(uint8_t mode)
```

Reserve the **xfer** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KXFERNOERR	The manager is reserved
		KXFERCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
...
status = xfer_reserve(KDEVALL);
...
while (xfer_reserve(KDEVALL) == KXFERCHBSY) {
    kern_switchFast();
}
xfer_xyz();
status = xfer_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(XFER, KDEVALL);
xfer_xyz();
RELEASE(XFER, KDEVALL);
```

```
int32_t xfer_release(uint8_t mode)
```

Release the **xfer** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KXFERNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = usdc_release(KDEVALL);
```

```
int32_t xfer_configure(cnfXfer_t *parameters)
```

Configure the **xfer** manager to operate. Before using the **xfer_startXfer** system calls it is mandatory to call this function.

input:	*parameters	Ptr on the xfer parameters
output:	status	KXFEROERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Supported structure elements:

```
struct cnfXfer {
    void        *oXferBuffer;      // Ptr on the xfer buffer
    uint32_t    oPacketSize;       // Size of a packet
    uint16_t    oNbPackets;        // Nb of packets
    void        (*oEndXfer)();     // Ptr on the callback EndXfer routine
};
```

Call example in C:

```
#define KNBCOLS      480          // Number of cols
#define KNBROWS       480          // Number of rows
#define KNBPACKETS    80           // Number of packs
#define KNBPACKMIO   (KNBPACKETS-1) // Number of packs minus 1
#define KPACKETSIZE ((KNBCOLS*KNBROWS)/KNBPACKETS) // Packet size

int32_t      status;
cnfxfer_t    configureXFER;
uint8_t      *image;
bool_t       imageReady = FALSE;
...
// Transfer buffer = image buffer, Size of the transfer
// Call back after the transfer of all the packets

img0_getImage((void *)&image);

configureXFER.oXferBuffer = image;
configureXFER.oPacketSize = KPACKETSIZE;
configureXFER.oNbPackets = KNBPACKETS;
configureXFER.oEndXfer   = _transferSPI;

status = xfer_configure(&configureXFER);
if (status != KXFERNOERR) iotx_printf(KSYST, "xfer manager problem\n");
...
// Start the first acquisition

img0_acquisition();
xfer_startXfer();
while (TRUE) {
    while (imageReady == FALSE) kern_switchFast();
    vImageReady = FALSE;

    img0_acquisition();
}
...
static void _transferSPI(void) {
    uint8_t      *image;

    img0_getImage((void *)&image);
    xfer_setXferBuffer(image);
    imageReady = TRUE;
}
```

int32_t xfer_setXferBuffer(void *xferBuffer)

Set the transfer buffer.

input:	*xferBuffer	Ptr on the xfer buffer
output:	status	KXFERNOERR OK
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
#define      KNBSECTORS      5;

      int32_t      status;
static uint8_t      buffer[752*480];
...
...
status = xfer_setXferBuffer(buffer);
```

```
int32_t xfer_startXfer(void)
```

Start the transfer.

input:	-	
output:	status	KXFERNOERR OK
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = xfer_startXfer();
```

int32_t xfer_stopXfer(void)

Stop the transfer.

input:	-	
output:	status	KXFERNOERR
reentrancy:	NO	OK

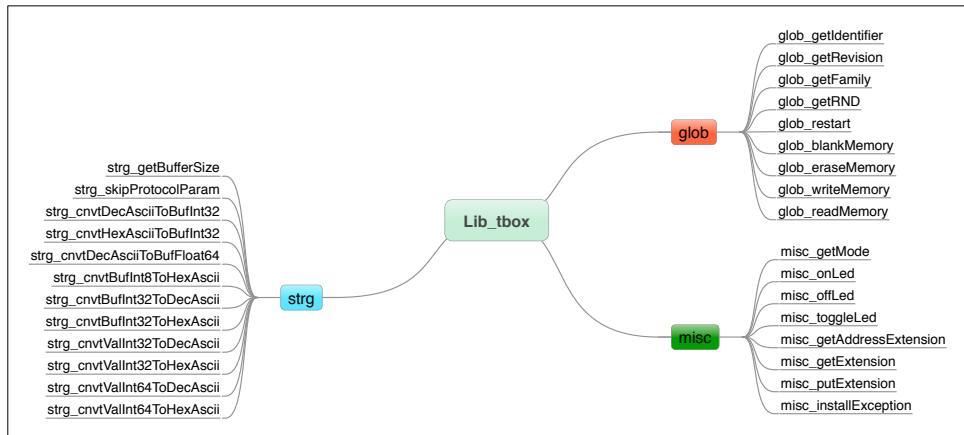
Use **RESERVE/RELEASE** macros
to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = xfer_stopXfer();
```

3.5. The lib_tbox library

This library contains the following managers, the **glob**, the **misc**, and the **strg**. Accordingly to the hardware that μKOS has to support, additional managers can be added. The **glob** is responsible for handling some generic functionalities. The **misc** is responsible for handling the extension bus as well as the exception management. The **strg** is a collection of functions for manipulating formatted strings and chars.



misc System Calls	
misc_getMode	Get the configuration mode
misc_onLed	Turn on a LED
misc_offLed	Turn off a LED
misc_toggleLed	Toggle the state of a LED
misc_muteLed	Mute the state of a LED
misc_getAddressExtension	Get the address of the extension bus
misc_getExtension	Get a byte from the relative address of the extension bus
misc_putExtension	Put a byte to the relative address of the extension bus
misc_installException	Install a routine for an exception

glob System Calls	
glob_getIdentifier	Get the system ASCII identifier Ptr
glob_getRevision	Get the TBOX revision
glob_getFamily	Get the Id of the system family
glob_getRND	Get a 32-bit random number
glob_restart	Restart the system
glob_blankMemory	Blank check of an arbitrary memory block size
glob_eraseMemory	Erase memory sectors
glob_writeMemory	Write an arbitrary memory block size
glob_readMemory	Read an arbitrary memory block size

strg System Calls	
strg_getBufferSize	Get the size of an ASCII buffer
strg_skipProtocolParam	Skip parameters from a formatted ASCII buffer (separator is ",")
strg_cnvtDecAsciiToBufInt32	Convert a formatted decimal ASCII buffer to a int32_t binary array
strg_cnvtHexAsciiToBufInt32	Convert a formatted hexadecimal ASCII buffer to a int32_t binary array
strg_cnvtDecAsciiToBufFloat64	Convert a formatted decimal ASCII buffer to a float64_t binary array
strg_cnvtBufBin8ToHexAscii	Convert a int8_t binary array to a formatted hexadecimal buffer
strg_cnvtBufInt32ToDecAscii	Convert a int32_t binary array to a formatted decimal buffer
strg_cnvtBufInt32ToHexAscii	Convert a int32_t binary array to a formatted hexadecimal buffer
strg_cnvtVallnt32ToDecAscii	Convert a Int32_t binary value to a decimal ASCII buffer
strg_cnvtVallnt32ToHexAscii	Convert a int32_t binary value to an hexadecimal ASCII buffer
strg_cnvtVallnt64ToDecAscii	Convert a int64_t binary value to a decimal ASCII buffer
strg_cnvtVallnt64ToHexAscii	Convert a int64_t binary value to an hexadecimal ASCII buffer

3.5.1. “glob” manager system calls

```
int32_t glob_getIdentifier(char_t **identifier)
```



Get the system ASCII identifier Ptr.

```
input:      **identifier          Ptr on the identifier Id
output:    status      KGLOBNOERR   OK
reentrancy: YES
```

Call example in C:

```
int32_t      status;
char_t       *identifier;
...
status = glob_getIdentifier(&identifier);
```

```
int32_t glob_getRevision(char_t **revision)
```

Get the TBox ASCII revision Ptr.

input:	**revision	Ptr on the TBox revision string
output:	status	KGLOBNOERR
reentrancy:	YES	OK

Call example in C:

```
int32_t      status;
char_t       *revision;
...
status = glob_getRevision(&revision);
```

```
int32_t glob_getFamily(uint32_t *family)
```

Get the Id of the system family.

input:	*family	Ptr on the family Id
output:	status	KGLOBNOERR OK
reentrancy:	YES	

Call example in C:

```
int32_t status;
uint32_t family;
...
status = glob_getFamily(&family);
```

```
int32_t glob_getRND(uint32_t *number)
```

Get a random number.

input:	*number	Ptr on the number
output:	status	KGLOBNOERR
	status	OK
		KGLOBSYCNA
reentrancy:	YES	System call not available

Call example in C:

```
int32_t      status;
uint32_t     number;
...
status = glob_getRND(&number);
```

int32_t glob_restart(void)

Restart the system.

```
input:      -
output:    status      KGLOBNOERR   OK
reentrancy: YES
```

Call example in C:

```
int32_t      status;
...
status = glob_restart();
```

```
int32_t glob_blankMemory(uint32_t address,
                         uint32_t size)
```

Blank check of an arbitrary memory region for Flash, EEPROM, other.

input:	address	Memory address (address or relative address of the device)
	size	Size in byte of the memory block
output:	status	KGLOBNOERR KGLOBFLNOE KGLOBSYCNA
		OK Memory segment not empty System call not available
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
uint32_t      size = 0x2000;
int32_t       status;
uint32_t      memory;
...
memory = 0x10000;

status = glob_blankMemory(memory, size);
```

```
int32_t glob_eraseMemory(uint32_t address,
                         uint32_t nbSectors)
```

Erase memory sectors for Flash, EEPROM, other.

input:	address	Memory address
	nbSectors	Number of sector to erase
output:	status	KGLOBNOERR
		KGLOBFLERA
		KGLOBSYCNA
reentrancy:	NO	System call not available Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
uint32_t      nbSectors = 3;
int32_t       status;
uint32_t      memory;
...
memory = 0x10000;

status = glob_eraseMemory(memory, nbSectors);
```

```
int32_t glob_writeMemory(uint832_t address,
                        const uint8_t *buffer,
                        uint32_t size)
```

Write memory sectors for Flash, EEPROM, other.

input:	address	Memory address
	*buffer	Ptr on the buffer (aligned at the beginning of the sector)
output:	size	Size in byte of the memory block
	status	KGLOBNOERR OK KGLOBFLWRT Memory segment not written KGLOBSYCNA System call not available
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
const uint8_t      source[KSIZE] = { 0,1,2,3,4,5,6,7,8,9 };
int32_t           status;
uint32_t          memory;

...
memory = 0x10000;

status = glob_writeMemory(memory, source, KSIZE);
```

```
int32_t glob_readMemory(uint32_t address,
                        uint8_t *buffer,
                        uint32_t size)
```

Read of an arbitrary memory region for Flash, EEPROM, other.

input:	address	Memory address
	*buffer	Ptr on the buffer
	size	Size in byte of the memory block
output:	status	KGLOBNOERR
		OK
		KGLOBFLWRT
		Memory segment not written
		KGLOBSYCNA
		System call not available
reentrancy:	NO	Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
uint8_t      destination[KSIZE];
int32_t      status;
uint32_t      memory;
...
memory = 0x10000;

status = glob_readMemory(memory, destination, KSIZE);
```

3.5.2. “misc” manager system calls

```
int32_t misc_getMode(uint32_t *mode)
```



Get the configuration mode (the system switches). The coding information returned by this system call depends on the target.

```
input:      *mode                  Ptr on the jumper value
output:     status        KMISCNOERR  OK
reentrancy: THREAD-SAFE
```

Call example in C:

```
int32_t      status;
uint32_t      mode;
...
status = misc_getMode(&mode);
```

```
int32_t misc_onLed(uint8_t ledNb)
```

Turn on a LED.

input:	ledNb	Led number
output:	status	KMISCNOERR OK
		KMISCNODEV The selected LED does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
uint32_t     ledNb;  
...  
status = misc_onLed(ledNb);
```

```
int32_t misc_offLed(uint8_t ledNb)
```

Turn off a LED.

input:	ledNb	Led number
output:	status	KMISCNOERR OK
		KMISCNODEV The selected LED does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
uint32_t     ledNb;  
...  
status = misc_offLed(ledNb);
```

int32_t misc_toggleLed(uint8_t ledNb)

Toggle the state of a LED.

input:	ledNb	Led number
output:	status	OK
	KMISCNOERR	The selected LED does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     ledNb;
...
status = misc_toggleLed(ledNb);
```

int32_t misc_muteLed(bool_t mute)

Mute the state of the LEDs.

input:	mute	TRUE	Turn off (mute) all the LEDs
		FALSE	Normal mode
output:	status	KMISCNOERR	OK
reentrancy:	THREAD-SAFE		

Call example in C:

```
int32_t      status;  
...  
status = misc_muteLed(TRUE);      //  
...  
...  
status = misc_muteLed(FALSE);    // Portion of code without any LED activity
```

```
int32_t misc_getAddressExtension(void **extension)
```

Get the address of the extension bus.

input:	**extension	Ptr on the extension bus
output:	status	KMISCNOERR OK
		KMISCSYCNA System call not available
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint8_t      *extension;
...
status = misc_getAddressExtension((void **) &extension);
*extension = 0x25;
```

```
int32_t misc_getExtension(uint8_t *byte,
                           uint8_t offset)
```

Get a byte from the relative address offset of bus.

input:	* byte	Ptr on the byte
input:	offset	Relative address offset
output:	status	KMISCNOERR OK
		KMISCSYCNA System call not available
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint8_t      byte;
...
status = misc_getExtension(&byte, 0x25);
```

```
int32_t misc_putExtension(uint8_t byte,
                           uint8_t offset)
```

Put a byte to the relative address offset of bus.

input:	byte	The byte
input:	offset	Relative address offset
output:	status	KMISCNOERR OK
		KMISCSYCNA System call not available
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint8_t      byte;
...
byte = 0x23;

status = misc_putExtension(byte, 0x87);
```

```
int32_t misc_installException(void (*code)(),
                             uint8_t exceptionNb)
```

Install a routine for an exception.

input:	*code	Ptr on the exception function
	exceptionNb	Exception number
output:	status	KMISCNOERR OK
		KMISCNODEV The selected exception does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
...
status = misc_installException(addressRoutine, 23);
```

3.5.3. “strg” manager system calls



```
int32_t strg_getBufferSize(const char_t *ascii,
                           uint32_t *size)
```

Get the size of an ASCII buffer.

```
(ASCII) -2345,2345,+6789CRLF0  out: (.32) 18
The \0 character is not counted.
```

input:	*ascii	Ptr on the ASCII buffer
	*size	Ptr on the size
output:	status	KSTRGNOERR OK
reentrancy:	YES	

Call example in C:

```
const char_t      ascii[] = "This is a string ...\\n\\r";
int32_t          status;
uint32_t          size;
...
status = strg_getBufferSize(ascii, &size);
```

```
int32_t strg_skipProtocolParam(char_t *ascii,
                               uint8_t nbParameters)
```

Skip parameters from a formatted ASCII buffer (**the separator is ','**).

input:	*ascii	Ptr on the ASCII buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR OK
		KSTRGNOPRM The number of parameters is = 0
		KSTRGTM prm Too many parameters to skip
reentrancy:	YES	

Call example in C:

```
int32_t      status;
char_t       ascii[] = "R,232,234,454,67567,7678\n\r";
...
status = strg_skipProtocolParam(ascii, 3);
```

```
int32_t strg_cnvtDecAsciiToBufInt32(int32_t *binary,
                                     const char_t *ascii,
                                     uint8_t nbParameters)
```

Convert a formatted decimal ASCII buffer to a int32_t binary array.

ASCII input buffer: int32_t output array:
(ASCII) -2345,2345,+6789CRLF (.32) -0x929 0x929 0x1A85

input:	*binary	Ptr on the binary buffer
	*ascii	Ptr on the ASCII buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR OK
		KSTRGNOPRM The number of parameters is = 0
		KSTRGNOASC The parameter is not an ASCII value
		KSTRGTODIG The parameter has too many digits
reentrancy:	YES	

Call example in C:

```
#define        KSIZE  10

const  char_t        ascii[] = "-2345,2345,+6789\n\r";
int32_t        status;
int32_t        binary[KSIZE];
...
status = strg_cnvtDecAsciiToBufInt32(binary, ascii, 3);
```

```
int32_t strg_cnvtHexAsciiToBufInt32(int32_t *binary,
                                      const char_t *ascii,
                                      uint8_t nbParameters)
```

Convert a formatted hexadecimal ASCII buffer to a int32_t binary array.

ASCII input buffer: int32_t output array:
(ASCII) -2345,2345,+1A85CRLF (.32) -0x2345 0x2345 0x1A85

input:	*binary	Ptr on the binary buffer
	*ascii	Ptr on the ASCII buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR OK
		KSTRGNOPRM The number of parameters is = 0
		KSTRGNOASC The parameter is not an ASCII value
		KSTRGTONDIG The parameter has too many digits
reentrancy:	YES	

Call example in C:

```
#define        KSIZE  5

const  char_t        ascii[] = "-2345,2345,+6789\n\r";
int32_t        status;
int32_t        binary[KSIZE];
...
status = strg_cnvtHexAsciiToBufInt32(binary, ascii, 3);
```

```
int32_t strg_cnvtDecAsciiToBufFloat64(float64_t *binary,
                                         const char_t *ascii,
                                         uint8_t nbParameters)
```

Convert a formatted decimal ASCII buffer to a float64_t binary array.

ASCII input buffer: float64_t output array:
(ASCII) -1e-2,2.98,+80.4e-5CRLF0 -1e-2 2.98 +80.4e-5

input:	*binary	Ptr on the binary buffer
	*ascii	Ptr on the ASCII buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR OK
		KSTRGNOPRM The number of parameters is = 0
		KSTRGNOASC The parameter is not an ASCII value
		KSTRGTODIG The parameter has too many digits
reentrancy:	YES	

Call example in C:

```
#define      KSIZE 3

const  char_t      ascii[] = "-1e-2,2.98,+80.4e-5\r\n";
int32_t      status;
float64_t     binary[KSIZE];
...
status = strg_cnvtDecAsciiToBufFloat64(binary, ascii, 3);
```

```
int32_t strg_cnvtBufInt8ToHexAscii(char_t *ascii,
                                    const int8_t *binary,
                                    uint8_t nbParameters)
```

Convert a int8_t binary array to a formatted hexadecimal ASCII buffer.

```
int8_t input array:          ASCII output buffer:  
(.8) -0x1 0x2 0xA0          ASCII) -1,2,+A0CRLF
```

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR OK
reentrancy:	YES	

Call example in C:

```
#define      KSIZE  256

const  int8_t      binary[] = { -0x1,0x2,0xA0 };
      int32_t     status;
      char_t       ascii[KSIZE];
...
status = strg_cnvtBufInt8ToHexAscii(ascii, binary, 3);
```

```
int32_t strg_cnvtBufInt32ToDecAscii(char_t *ascii,
                                     const int32_t *binary,
                                     uint8_t nbParameters)
```

Convert a int32_t binary array to a formatted decimal ASCII buffer.

```
int32_t input array:          ASCII output buffer:  
(.32) -0x929 0x929 0x1A85  (ASCII) -2345,2345,+6789CRLF
```

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR OK
reentrancy:	YES	

Call example in C:

```
#define      KSIZE   256  
  
const  int32_t      binary[] = { -0x929,0x929,0x1A85 };  
      int32_t      status;  
      char_t       ascii[KSIZE];  
...  
status = strg_cnvtBufInt32ToDecAscii(ascii, binary, 3);
```

```
int32_t strg_cnvtBufInt32ToHexAscii(char_t *ascii,
                                     const int32_t *binary,
                                     uint8_t nbParameters)
```

Convert a int32_t binary array to a formatted hexadecimal ASCII buffer.

```
int32_t input array:          ASCII output buffer:  
(.32) -0x2345 0x2345 0x1A85      (ASCII) -2345,2345,+1A85CRLF
```

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary buffer
	nbParameters	Nb. of parameters
output:	status	KSTRGNOERR
reentrancy:	YES	OK

Call example in C:

```
#define      KSIZE   256

const  int32_t      binary[] = { -0x929,0x929,0x1A85 };
int32_t      status;
char_t       ascii[KSIZE];
...
status = strg_cnvtBufInt32ToHexAscii(ascii, binary, 3);
```

```
int32_t strg_cnvtValInt32ToDecAscii(char_t *ascii,
                                     const int32_t *binary,
                                     char_t **buffer)
```

Convert a int32_t binary value to a decimal ASCII buffer.

```
int32_t input value:          ASCII output buffer:  
(.32) -0x929      (ASCII) -2345
```

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary value
	**buffer	Ptr on the first available position of the ASCII buffer
output:	status	KSTRGNOERR
reentrancy:	YES	OK

Call example in C:

```
#define      KSIZE  256

int32_t      status;
int32_t      value = -233;
char_t       ascii[KSIZE], *wkAscii;
...
status = strg_cnvtValInt32ToDecAscii(ascii, &value, &wkAscii);
```

```
int32_t strg_cnvtValInt32ToHexAscii(char_t *ascii,
                                     const int32_t *binary,
                                     char_t **buffer)
```

Convert a int32_t binary value to a hexadecimal ASCII buffer.

int32_t input value: ASCII output buffer:
(.32) -0x92A (ASCII) -0x92A

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary value
	**buffer	Ptr on the first available position of the ASCII buffer
output:	status	KSTRGNOERR
reentrancy:	YES	OK

Call example in C:

```
#define        KSIZE  256

int32_t      status;
int32_t      value = -233;
char_t        ascii[KSIZE], *wkAscii;
...
status = strg_cnvtValInt32ToHexAscii(ascii, &value, &wkAscii);
```

```
int32_t strg_cnvtValInt64ToDecAscii(char_t *ascii,
                                     const int64_t *binary,
                                     char_t **buffer)
```

Convert a int64_t binary value to a decimal ASCII buffer.

```
int64_t input value:          ASCII output buffer:  
(.64) -0x929                (ASCII) -2345
```

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary value
	**buffer	Ptr on the first available position of the ASCII buffer
output:	status	KSTRGNOERR
reentrancy:	YES	OK

Call example in C:

```
#define      KSIZE 256

int32_t      status;
int64_t      value = -233;
char_t       ascii[KSIZE], *wkAscii;
...
status = strg_cnvtValInt64ToDecAscii(ascii, &value, &wkAscii);
```

```
int32_t strg_cnvtValInt64ToHexAscii(char_t *ascii,
                                     const int64_t *binary,
                                     char_t **buffer)
```

Convert a int64_t binary value to a hexadecimal ASCII buffer.

```
int64_t input value:          ASCII output buffer:  
(.64) -0x92A                (ASCII) -0x92A
```

input:	*ascii	Ptr on the ASCII buffer
	*binary	Ptr on the binary value
	**buffer	Ptr on the first available position of the ASCII buffer
output:	status	KSTRGNOERR
reentrancy:	YES	OK

Call example in C:

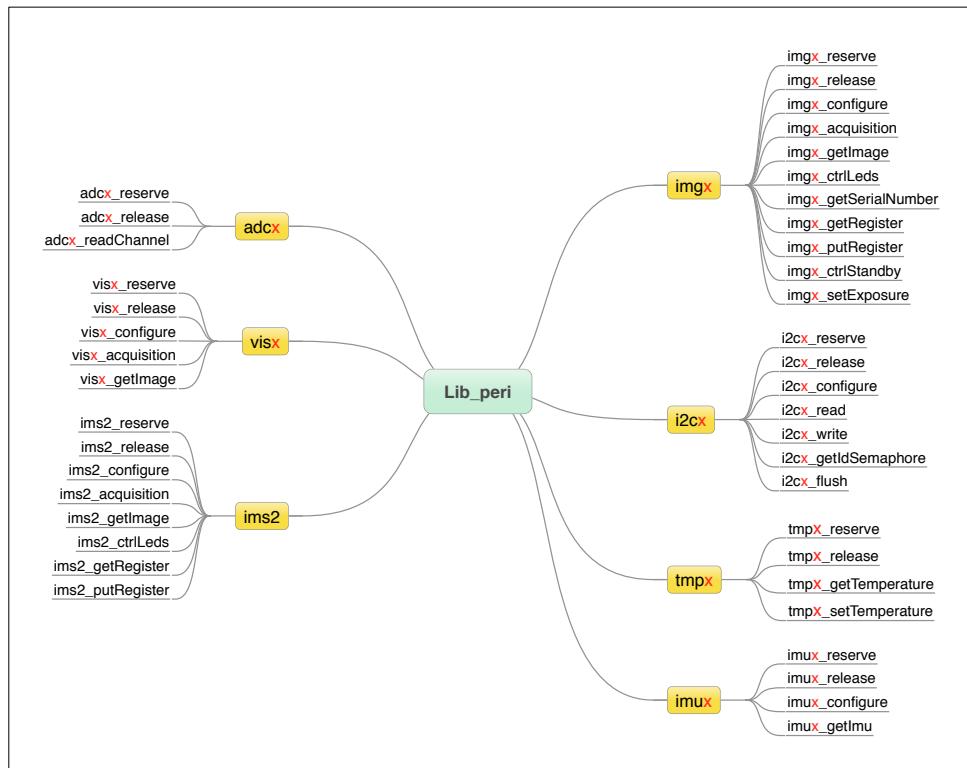
```
#define      KSIZE  256

int32_t      status;
int64_t      value = -233;
char_t       ascii[KSIZE], *wkAscii;
...
status = strg_cnvtValInt64ToHexAscii(ascii, &value, &wkAscii);
```

3.6. The lib_peri library

This library contains the following managers, **imgx**, **ims2**, **visx**, **tmpx**, **imu0**, **i2cx**, and the **adcx** one. Accordingly to the hardware that μKOS has to support, additional managers can be implemented.

All the managers that control similar hardware should have the same set system calls and the same I/O parameters (i.e. vis0, vis1, img0, img1, etc.). The configuration is done via a the manager specific structures.



visx System Calls	
visx_reserve	Reserve the visx manager
visx_release	Release the visx manager
visx_configure	Configure the visx manager
visx_acquisition	Acquire an image on the visx manager
visx_getImage	Get an image ptr on the visx manager

imgx System Calls	
imgx_reserve	Reserve the imgx manager
imgx_release	Release the imgx manager
imgx_configure	Configure the imgx manager
imgx_acquisition	Acquire an image on the imgx manager
imgx_getImage	Get an image ptr on the imgx manager
imgx_ctrlLeds	Control of the LEDs
imgx_getSerialNumber	Get the factory serial number of the device
imgx_getRegister	Get the value of an imager register
imgx_putRegister	Put a value into an imager register
imgx_ctrlStandby	Control of the standby mode of the imager (only in KSNAP mode)
imgx_setExposure	Set the exposure time

tmpx System Calls	
tmpx_reserve	Reserve the tmpx manager
tmpx_release	Release the tmpx manager
tmpx_getTemperature	Get the temperature
tmpx_setTemperature	Set the temperature

adcx System Calls	
adcx_reserve	Reserve the adcx manager
adcx_release	Release the adcx manager
adcx_getAnalog	get an analog value

i2cx System Calls	
i2cx_reserve	Reserve the i2cx manager
i2cx_release	Release the i2cx manager
i2cx_configure	Configure the i2cx manager
i2cx_read	Read a buffer
i2cx_write	Write a buffer

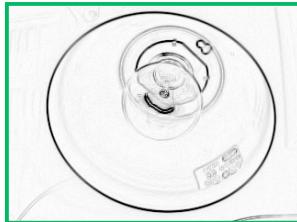
imux System Calls	
imux_reserve	Reserve the imux manager
imux_release	Release the imux manager
imux_configure	Configure the imux manager
imux_getimu	Get the IMU data (accelerometer, gyroscope, magnetometer)

ims2 System Calls	
ims2_reserve	Reserve the ims2 manager
ims2_release	Release the ims2 manager
ims2_configure	Configure the ims2 manager
ims2_acquisition	Acquire an image on the ims2 manager
ims2_getImage	Get an image ptr on the ims2 manager
ims2_ctrlLeds	Control of the LEDs
ims2_getRegister	Get the value of a S2 register
ims2_putRegister	Put a value into a S2 register

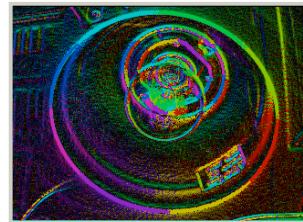
3.6.1. “visx” manager system calls



Luminance



Contrast



Oricon

int32_t visx_reserve(uint8_t mode)

Reserve the **visx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KVISXNOERR	The manager is reserved
		KVISXCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
...
status = visx_reserve(KDEVALL);
...
while (visx_reserve(KDEVALL) == KVISXCHBSY) {
    kern_switchFast();
}
visx_xyz();
status = visx_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(VISX, KDEVALL);
visx_xyz();
RELEASE(VISX, KDEVALL);
```

```
int32_t visx_release(uint8_t mode)
```

Release the **visx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KVISXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = visx_release(KDEVALL);
```

```
int32_t visx_configure(const cnfVisx_t *configure)
```

Configure the **visx** manager. The semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

```
"visx - img OK"      End of the acquisition
```

input:	*configure	Ptr on the configuration information
output:	status	KVISXNOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Supported structure elements:

```
struct cnfVisx {
    uint8_t     oImager;           // Imager
    #define KLOG      0             // Logarithmic imager
    #define KLIN      1             // Linear imager
    #define KLINLOG   2             // LinLog imager

    uint8_t     oCommand;          // Command
    #define KLUM10    0             // Integrate an 10-bit luminance image
    #define KLUM8MSB  1             // Integrate an 8-bit (MSB) luminance image
    #define KLUM8LSB  2             // Integrate an 8-bit (LSB) luminance image
    #define KCONTRAST 3             // Integrate a contrast image
    #define KORICON   4             // Integrate an oricon image

    uint8_t     oAcqPage;          // Acquisition page
    #define KPAGE0   0              // Acquisition & transfer on SDRAM the page 0
    #define KPAGE1   1              // Acquisition & transfer on SDRAM the page 1
    #define KPPI     2              // Acquisition & transfer on the PPI
    #define KNOXFER  3              // Acquisition & no transfer

    uint16_t    oExposureTime;     // Exposure time
    uint8_t     oVrefRange;         // DAC8 VRef range

    uint8_t     oKernSync;          // Kernel synchro
    #define BIMSEMA  0              // IM semaphore
};

};
```

Call example in C:

```
int32_t      status;
const cnfVisx_t  configure = {
    .oCommand     = KLUM8MSB;
    .oAcqPage    = KPAGE0;
    .oKernSync   = (1<<BIMSEMA);
};

...
status = visx_configure(&configure);
```

int32_t visx_acquisition(void)

Acquisition of an image on the **visx** manager. This call launches all the necessary phases to integrate an image and to transfer it into the SDRAM.

input:	-		
output:	status	KVISXNOERR	OK
		KVISXTROPR	Acquisition/transfer on progress
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = visx_acquisition();
```

```
int32_t visx_getImage(uint8_t page,
                      void **image)
```

Get the image pointer.

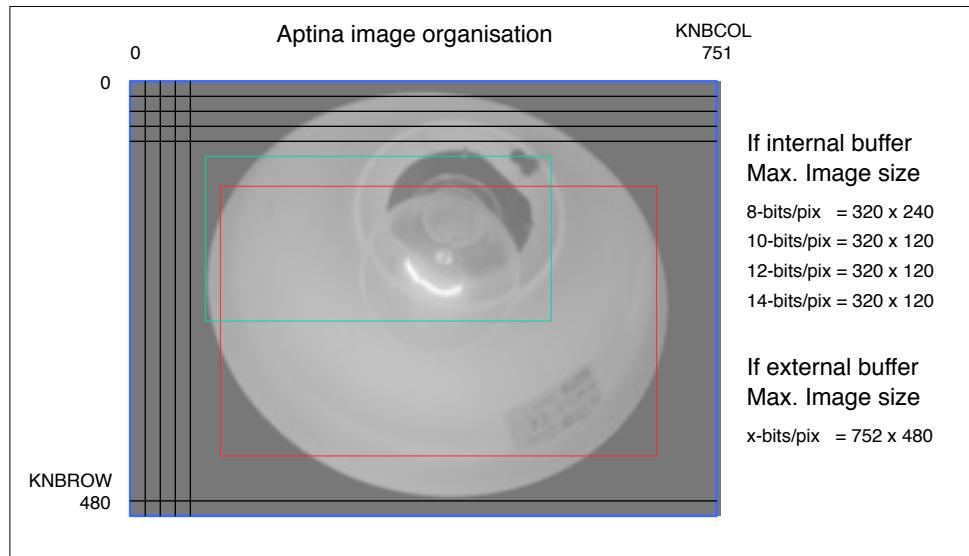
input:	page	Page number [KPAGE0, KPAGE1, KPPI, KNOXFER]
	**image	Ptr to an image
output:	status	KVISXNOERR OK
		KVISXPGNOE The image page does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Notice: for hardware compatibility it should avoid the usage of the SDRAM (KPAGE0 and KPAGE1)

Call example in C:

```
int32_t      status;
uint8_t      *image;
...
status = visx_getImage(KPAGO, &image);
```

3.6.2. “imgx” manager system calls



```
int32_t imgx_reserve(uint8_t mode)
```

Reserve the **imgx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KIMGXNOERR	The manager is reserved
		KIMGXCHBSY	The manager is busy
		KIMGXTIMEO	Timeout error
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = imgx_reserve(KDEVALL);  
...  
while (imgx_reserve(KDEVALL) == KIMGXCHBSY) {  
    kern_switchFast();  
}  
imgx_xyz();  
status = imgx_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(imgx, KDEVALL);  
imgx_xyz();  
RELEASE(imgx, KDEVALL);
```

```
int32_t imgx_release(uint8_t mode)
```

Release the **imgx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KIMGXNOERR	OK
		KIMGXTIMEO	Timeout error
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = imgx_release(KDEVALL);
```

```
int32_t imgx_configure(const cnfImgx_t *configure)
```

Configure the **imgx** manager. The semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

```
"imgx - img OK"      End of the acquisition  
"imgx - V synchro"   Vertical synchro
```

input:	*configure	Ptr on the configuration information
output:	status	KIMGXNOERR
		OK
		KIMGXTIMEO
		Timeout error
		KIMGXNOMEM
		Not enough memory
reentrancy:	NO	Use RESERVE/RELEASE macros
		to be thread-safe

Important:

1. The size of the oNbRows and oNbCols have to be a **multiple of 4**.

Available examples:

1. oAcqPage = KINTERNAL, oNbRows = 60, oNbCols = 60.
2. oAcqPage = KINTERNAL, oNbRows = 240, oNbCols = 320.
3. oAcqPage = KINTERNAL, oNbRows = 512, oNbCols = 248.
4. oAcqPage = KINTERNAL, oNbRows = 360, oNbCols = 360.
5. oAcqPage = KEXTERNAL, oNbRows = 480, oNbCols = 752.

Supported structure elements:

```
struct cnfImgx {
    uint8_t      oAcqMode           // Acquisition mode
    #define      KCONT              0   // KCONT = continuous mode
    #define      KSNAP              1   // KSNAP = snapshot mode
    #define      KSYNC              2   // KSYNC = sync mode

    uint8_t      oPixMode;          // Pixel mode
    #define      PIX8BITS           0   // PIX8BITS = 8-bit resolution
    #define      PIX10BITS          1   // PIX10BITS = 10-bit resolution
    #define      PIX12BITS          2   // PIX12BITS = 12-bit resolution
    #define      PIX14BITS          3   // PIX14BITS = 14-bit resolution

    uint8_t      oAcqPage;          // Acquisition storage page or device
    #define      KINTERNAL           0   // Acquisition result on the internal memory
    #define      KEXTERNAL           1   // Acquisition result on the external memory

    uint16_t     oStRows;           // Start of rows
    uint16_t     oNbRows;           // Number of rows (has to be multiple of 4)
    #define      KNBROWSQVGVA       240 // Number of rows (default)

    uint16_t     oStCols;           // Start of cols
    uint16_t     oNbCols;           // Number of cols (has to be multiple of 4)
    #define      KNBCOLSQVGVA       320 // Number of columns (default)

    uint8_t      oKernSync;         // uKernel synchro
    #define      BIMSEMA             0   // IM semaphore
    #define      BVSSEMA             1   // Vertical synchro semaphore

    void        *oImgCnf;          // Imager configuration table

    void        (*oHSync)(void);    // Ptr on the callback HSync routine
    void        (*oFrame)(void);    // Ptr on the callback Frame routine
    void        (*oVSync)(void);    // Ptr on the callback VSync routine
    void        (*oDMAEc)(void);   // Ptr on the callback DMAEc routine
};

};
```

Call example in C:

```
// Image 60 x 60 pixels; this table concerns the Aptina MT9V032 imager
// Fixed exposure

static const mt9v032_t      aTab060060F[] = {
    { 1,    0x0001 }, { 2,    0x0005 }, { 3,    0x003C }, { 4,    0x003D },
    { 5,    0x026F }, { 6,    0x0032 }, { 7,    0x0388 }, { 8,    0x01E0 },
    { 9,    0x3FFE }, { 10,   0x0164 }, { 11,   0x001D }, { 16,   0x002D },
    { 21,   0x8032 }, { 32,   0x03D5 }, { 33,   0x0018 }, { 34,   0x0028 },
    { 40,   0x0016 }, { 41,   0x001C }, { 43,   0x0003 }, { 44,   0x0006 },
    { 46,   0x0004 }, { 47,   0x0003 }, { 70,   0x1606 }, { 71,   0x8081 },
    { 72,   0x007F }, { 104,  0xE5F9 }, { 105,  0x65D7 }, { 106,  0x2950 },
    { 107,  0xA040 }, { 112,  0x0032 }, { 114,  0x0001 }, { 115,  0x0307 },
    { 116,  0x0010 }, { 128,  0x00F4 }, { 129,  0x0004 }, { 130,  0x0004 },
    { 131,  0x0004 }, { 132,  0x0004 }, { 133,  0x0004 }, { 134,  0x0004 },
    { 135,  0x0004 }, { 136,  0x0004 }, { 137,  0x0004 }, { 138,  0x0004 },
    { 139,  0x0004 }, { 140,  0x0004 }, { 141,  0x0004 }, { 142,  0x0004 },
    { 143,  0x0004 }, { 144,  0x0004 }, { 145,  0x0004 }, { 146,  0x0004 },
    { 147,  0x0004 }, { 148,  0x0004 }, { 149,  0x0004 }, { 150,  0x0004 },
    { 151,  0x0004 }, { 152,  0x0004 }, { 154,  0x003B }, { 155,  0x02F0 },
    { 156,  0x02F0 }, { 157,  0x02F0 }, { 160,  0x003B }, { 161,  0x01E0 },
    { 162,  0x01E0 }, { 163,  0x01E0 }, { 165,  0x0020 }, { 166,  0x0008 },
    { 168,  0x0001 }, { 169,  0x0008 }, { 171,  0x0001 }, { 175,  0x0000 },
    { 176,  0x01B0 }, { 189,  0x006C }, { 194,  0x0850 }, { 0,    0x0000 }
};

int32_t      status;
const cnfImgx_t  configure = {
    .oAcqMode      = KCONT,
    .oPixMode       = PIX8BITS,
    .oAcqPage       = KINTERNAL,
    .oNbRows        = 60,
    .oNbCols        = 60,
    .oKernSync      = (1<<BIMSEMA) | (1<<BVSSSEMA),
    .oHSync          = 0;
    .oFrame          = 0;
    .oVSync          = 0;
    .oDMAEC         = _myRoutine;
    .oImgCnf         = (void *)aTab060060F,
};

...
status = imgx_configure(&configure);

static void  _myRoutine(void) {
...
}
```

int32_t imgx_acquisition(void)

Acquire an image into the memory (internal or external). This call launches all the necessary phases to integrate an image and to transfer it into the RAM.

input:	-		
output:	status	KIMGXNOERR	OK
		KIMGXTIMEO	Timeout error
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = imgx_acquisition();
```

```
int32_t imgx_getImage(void **image)
```

Get the image pointer.

input:	**image	Ptr to an image
output:	status	KIMGXNOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
int32_t      status;
uint8_t      *image;
...
status = imgx_getImage(&image);
```

```
int32_t imgx_ctrlLeds(uint8_t ledStates)
```

Control of the LEDs.

input:	ledState	State of the LEDs (0 = off, 1 = on)
output:	status	KIMGXNOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = imgx_ctrlLeds(0b00001010);
```

```
int32_t imgx_getSerialNumber(uint64_t *serialNumber)
```

Get the factory serial number of the device.

input:	*serialNumber	Ptr on the serial number
output:	status	KIMGxNOERR OK
		KIMGxTIMEO Timeout error
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
uint64_t     serialNumber;
...
status = imgx_getSerialNumber(&serialNumber);
```

```
int32_t imgx_getRegister(uint8_t registerNb,  
                         uint16_t *value)
```

Get the value of an imager register.

input:	registerNb	Imager register number
	*value	Ptr on the value
output:	status	KIMGXNOERR OK
		KIMGXTIMEO Timeout error
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
uint16_t     value;  
...  
status = imgx_getRegister(0x34, &value);
```

```
int32_t imgx_putRegister(uint8_t registerNb,  
                         uint16_t value)
```

Put a value into an imager register.

input:	registerNb	Imager register number
	value	The value
output:	status	KIMGXNOERR OK
		KIMGXTIMEO Timeout error
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
uint16_t     value = 0x1234;  
...  
status = imgx_putRegister(0x34, value);
```

int32_t imgx_ctrlStandby(uint8_t mode)

Control of the standby mode of the imager (only in KSNAP mode).

input:	mode	KOPERATE	Standby off (running mode)
	mode	KSTANDBY	Standby on
output:	status	KIMGxNOERR	OK
		KIMGxBDMOD	Bad mode for called function
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = imgx_ctrlStandby(KSTANDBY);
```

int32_t imgx_setExposure(uint32_t time)

Set the exposure time of the imager in μS.

input:	time	Exposure time. -1 force the mode
		Auto-exposure
output:	status	KIMGXNOERR OK
		KIMGXBDMOD Bad mode for called function
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = imgx_setExposure(15000);
```

3.6.3. “adc~~x~~” manager system calls

```
int32_t adcx_reserve(uint8_t mode)
```



Reserve the **adc~~x~~** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KADC x NOERR	The manager is reserved
		KADC x CHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = adcx_reserve(KDEVALL);  
...  
while (adcx_reserve(KDEVALL) == KADCxCHBSY) {  
    kern_switchFast();  
}  
adcx_xyz();  
status = adcx_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(ADCx, KDEVALL);  
adcx_xyz();  
RELEASE(ADCx, KDEVALL);
```

```
int32_t adcx_release(uint8_t mode)
```

Release the **adcx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KADCXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = adcx_release(KDEVALL);
```

```
int32_t adcx_getAnalog(uint8_t channelNb,
                       float64_t *reference,
                       float64_t *data)
```

Get an analog value.

input:	channelNb	Channel number
	*reference	Ptr on the ADC reference
	*data	Ptr on the data (result of the conversion)
output:	status	KADCXNOERR OK
		KADCXNODEV The channel does not exist
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
float64_t    reference, result[6];
uint8_t      channel;
...
for (channel = 0; channel < 6; channel++) {
    status = adcx_getAnalog(channel, &reference, &result[channel]);
    if (status != KADCXNOERR) {
        return (status);
    }
}
```

3.6.4. “tmpx” manager system calls



```
int32_t tmpx_reserve(uint8_t mode)
```

Reserve the **tmpx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KTMPXNOERR	The manager is reserved
		KTMPXCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = tmpx_reserve(KDEVALL);  
...  
while (tmpx_reserve(KDEVALL) == KTMPXCHBSY) {  
    kern_switchFast();  
}  
tmpx_xyz();  
status = tmpx_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(TMPX, KDEVALL);  
tmpx_xyz();  
RELEASE(TMPX, KDEVALL);
```

```
int32_t tmpx_release(uint8_t mode)
```

Release the **tmpx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KTMPXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = tmpx_release(KDEVALL);
```

```
int32_t tmpx_getTemperature(float64_t *temperature)
```

Get the temperature (represented in degrees [C] and in float64_t.

input:	*temperature	Ptr on the temperature
output:	status	KTMPONOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
int32_t      status;
float64_t    temperature;
uint16_t     temperatureInt, temperatureFrc;
...
status = tmpx_getTemperature(&temperature);
temperatureInt = (uint16_t)temperature;
temperatureFrc = (uint16_t)((temperature - (float64_t)(temperatureInt)) * 100);
iotx_printf(KSYST, "Temp = %d.%d\n", temperatureInt, temperatureFrc);
```

int32_t tmpx_setTemperature(float64_t temperature)

Set the temperature (represented in degrees [C] and in float64_t.

input:	temperature	Temperature
output:	status	KTMPONOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
int32_t      status;
float64_t    temperature = 37.6;
...
status = tmpx_setTemperature(temperature);
```

3.6.5. “i2cx” manager system calls

```
int32_t i2cx_reserve(uint8_t mode)
```



Reserve the **i2cx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KI2CXNOERR	The manager is reserved
		KI2CXCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = i2cx_reserve(KDEVALL);  
...  
while (i2cx_reserve(KDEVALL) == KI2CXCHBSY) {  
    kern_switchFast();  
}  
i2cx_xyz();  
status = i2cx_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(I2CX, KDEVALL);  
i2cx_xyz();  
RELEASE(I2CX, KDEVALL);
```

```
int32_t i2cx_release(uint8_t mode)
```

Release the **i2cx** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KI2CXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = i2cx_release(KDEVALL);
```

```
int32_t i2cx_configure(const cnfI2cx_t *configure)
```

Configure the **i2cx** manager. Semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

```
"i2cx - RX msg"      Buffer read  
"i2cx - TX msg"      Buffer written
```

input:	*configure	Ptr on the configuration information
output:	status	KI2CXNOERR
reentrancy:	NO	OK Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct cnfI2cx {  
    uint8_t      oSpeed          // Speed  
    #define      K100KHZ         0  // Bus at 100-KHz  
    #define      K400KHZ         1  // Bus at 400-KHz  
  
    uint8_t      oKernSync;      // uKernel synchro  
    #define      BRXSEMA        0  // RX semaphore  
    #define      BTXSEMA        1  // TX semaphore  
};
```

Call example in C:

```
int32_t      status;  
const cnfI2cx_t configure = {  
    .oSpeed      = K100KHZ,  
    .oKernSync   = (1<<BRXSEMA) | (1<<BTXSEMA),  
};  
...  
status = i2cx_configure(&configure);
```

```
int32_t i2cx_write(uint8_t deviceAdd,
                    const uint8_t *buffer,
                    uint16_t size)
```

Write a buffer to the **i2cx** device.

input:	deviceAdd	i2cx device address
	*buffer	Ptr on the buffer
	size	Size of the buffer
output:	status	KI2CXNOERR OK
		KI2CXTIMEO Timeout error
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
const uint8_t    buffer[7] = { 1, 2, 3, 4, 5, 6, 7 };
...
status = i2cx_write(0x34, buffer, sizeof(buffer));
```

```
int32_t i2cx_read(uint8_t deviceAdd,
                   uint8_t *buffer,
                   uint16_t size)
```

Read a buffer from the **i2cx** device.

input:	deviceAdd	i2cx device address
	*buffer	Ptr on the buffer
		If buffer[0] != 0xFF, then write the register number contained in the buffer[0] before reading the device
		If buffer[0] == 0xFF, then read the device
	size	Size of the buffer
output:	status	KI2CXNOERR
		OK
		KI2CXTIMEO
reentrancy:	NO	Timeout error
		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
uint8_t      buffer[10];
...
// Example 1: Read 10 bytes

buffer [0] = 0xFF;
status = i2cx_read(0x34, buffer, 10);

// Example 2: Write the register 56 and then read 2 bytes

buffer [0] = 56;
status = i2cx_read(0x34, buffer, 2);
```

`int32_t i2cx_flush(void)`

Flush the **i2cx** manager.

input:	-	
output:	status	KI2CXNOERR OK
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = i2cx_flush();
```

```
int32_t i2cx_getIdSemaphore(uint8_t semaphore,
                            char_t **identifier)
```

Get the semaphore identifier of the selected RX-TX **i2cx** manager.

input:	semaphore	KSEMARX	Select the RX semaphore
		KSEMATX	Select the TX semaphore
	**identifier		Ptr on the semaphore Id
output:	status	KI2CXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;
char_t       *identifier[2];
...
status = i2cx_getIdSemaphore(KSEMARX, &identifier[0]);
status = i2cx_getIdSemaphore(KSEMATX, &identifier[1]);
iotx_printf(KSYST, "Semaphore ids: %s, ...%s\n", identifier[0], identifier[1]);
```

3.6.6. “imux” manager system calls

```
int32_t imux_reserve(uint8_t mode)
```



Reserve the **imux** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KIMUXNOERR	The manager is reserved
		KIMUXCHBSY	The manager is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = imux_reserve(KDEVALL);  
...  
while (imux_reserve(KDEVALL) == KIMUXCHBSY) {  
    kern_switchFast();  
}  
imux_xyz();  
status = imux_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(IMUX, KDEVALL);  
imux_xyz();  
RELEASE(IMUX, KDEVALL);
```

```
int32_t imux_release(uint8_t mode)
```

Release the **imux** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KIMUXNOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = imux_release(KDEVALL);
```

```
int32_t imux_configure(const cnfImux_t *configure)
```

Configure the **imux** manager.

input:	*configure	Ptr on the configuration information
output:	status	KIMUXNOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Supported structure elements:

```
struct cnfImux {  
    uint8_t     oAcceMode;           // Acceleration mode  
    #define      K2G      0           // +/- 2-g  
    #define      K4G      1           // +/- 4-g  
    #define      K8G      2           // +/- 8-g  
    #define      K16G     3           // +/- 16-g  
  
    uint8_t     oGyroMode;          // Gyroscope mode  
    #define      K245DPS 0           // +/- 245-dps  
    #define      K500DPS 1           // +/- 500-dps  
    #define      K2000DPS 2          // +/- 2000-dps  
  
    uint8_t     oMagnMode;          // Magnetometer mode  
    #define      K4GAUSS 0           // +/- 4-gauss  
    #define      K8GAUSS 1           // +/- 8-gauss  
    #define      K12GAUSS 2          // +/- 12-gauss  
    #define      K16GAUSS 3          // +/- 16-gauss  
};
```

Supported structure elements:

```
struct accePack {
    float64_t    oAcce_X;           // X Accelerometer
    float64_t    oAcce_Y;           // Y Accelerometer
    float64_t    oAcce_Z;           // Z Accelerometer
};

struct gyroPack {
    float64_t    oGyro_X;          // X gyroscope
    float64_t    oGyro_Y;          // Y gyroscope
    float64_t    oGyro_Z;          // Z gyroscope
};

struct magnPack {
    float64_t    oMagn_X;          // X magnetometer
    float64_t    oMagn_Y;          // Y magnetometer
    float64_t    oMagn_Z;          // Z magnetometer
};
```

Call example in C:

```
int32_t      status;
const cnfImux_t  configure = {
    .oAcceMode    = K4G;
    .oGyroMode    = K2000DPS;
    .oMagnMode    = K8GAUSS;
};

...
status = imuX_configure(&configure);
```

```
int32_t imux_getImu(accePack_t *accelerometer,
                     gyroPack_t *gyroscope,
                     magnPack_t *magnetometer)
```

Get the **imux** data.

input:	*accelerometer	Ptr on the accelerometer pack
	*gyroscope	Ptr on the gyroscope pack
	*magnetometer	Ptr on the magnetometer pack
output:	status	KIMUXNOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

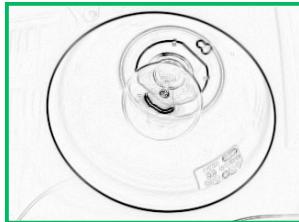
Call example in C:

```
int32_t      status;
accePack_t   accelerometer;
gyroPack_t   gyroscope;
magnPack_t   magnetometer;
...
status = imux_getImu(&accelerometer, &gyroscope, &magnetometer);
printf("%3f %3f %3f,    %3f %3f %3f,    %3f %3f %3f\n", \
       accelerometer.oAcce_X, \
       accelerometer.oAcce_Y, \
       accelerometer.oAcce_Z, \
       gyroscope.oGyro_X, \
       gyroscope.oGyro_Y, \
       gyroscope.oGyro_Z, \
       magnetometer.oMagn_X, \
       magnetometer.oMagn_Y, \
       magnetometer.oMagn_Z);
```

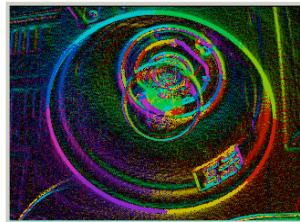
3.6.7. “ims2” manager system calls



Luminance



Contrast



Oricon

int32_t ims2_reserve(uint8_t mode)

Reserve the **ims2** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KIMS2NOERR	The manager is reserved
		KIMS2CHBSY	The manager is busy
		KIMS2TIMEO	Timeout error
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = ims2_reserve(KDEVALL);  
...  
while (ims2_reserve(KDEVALL) == KIMS2CHBSY) {  
    kern_switchFast();  
}  
ims2_xyz();  
status = ims2_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(IMS2, KDEVALL);  
ims2_xyz();  
RELEASE(IMS2, KDEVALL);
```

```
int32_t ims2_release(uint8_t mode)
```

Release the **ims2** manager.

input:	mode	KDEVALL	For read and write operations
output:	status	KIMS2NOERR	OK
		KIMS2TIMEO	Timeout error
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = ims2_release(KDEVALL);
```

```
int32_t imS2_configure(const cnfIms2_t *configure)
```

Configure the **imS2** manager. The semaphores can be activated for synchronizing processes with the manager events. Here are the available semaphores:

```
"imS2 - img OK"      End of the acquisition  
"imS2 - V synchro"   Vertical synchro
```

input:	*configure	Ptr on the configuration information
output:	status	KIMS2NOERR OK
		KIMS2TIMEO Timeout error
		KIMS2NOMEM Not enough memory
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Important:

1. The size of the oNbRows and oNbCols have to be a **multiple of 4**.

Available examples:

1. oAcqPage = KINTERNAL, oNbRows = 60, oNbCols = 60.
2. oAcqPage = KINTERNAL, oNbRows = 240, oNbCols = 320.
3. oAcqPage = KINTERNAL, oNbRows = 512, oNbCols = 248.

Supported structure elements:

```
struct cnfIms2 {
    uint8_t      oAcqMode           // Acquisition mode
    #define      KCONT              0   // KCONT = continuous mode
    #define      KSNAP              1   // KSNAP = snapshot mode
    #define      KSYNC              2   // KSYNC = sync mode

    uint8_t      oPixMode;          // Pixel mode
    #define      KLUMINANCE8        0   // KLUMINANCE 8-bits
    #define      KLUMINANCE10       1   // KLUMINANCE 10-bits
    #define      KCONTRAST8         2   // KCONTRAST 8-bits
    #define      KCONTRAST10        3   // KCONTRAST 10-bits
    #define      KORICON8           4   // KORICON 8-bits
    #define      KORICON10          5   // KORICON 10-bits

    uint8_t      oAcqPage;          // Acquisition storage page or device
    #define      KINTERNAL           0   // Acquisition result on the internal memory
    #define      KEXTERNAL           1   // Acquisition result on the external memory

    uint16_t     oStRows;           // Start of rows
    uint16_t     oNbRows;           // Number of rows (has to be multiple of 4)
    #define      KNBROWSQVGA        240 // Number of rows (default)

    uint16_t     oStCols;           // Start of cols
    uint16_t     oNbCols;           // Number of cols (has to be multiple of 4)
    #define      KNBCOLSQVGA        320 // Number of columns (default)

    uint8_t      oKernSync;         // uKernel synchro
    #define      BIMSEMA             0   // IM semaphore
    #define      BVSSEMA             1   // Vertical synchro semaphore

    void         *oImgCnf;          // Imager configuration table
    void         (*oHSync)(void);    // Ptr on the callback HSync routine
    void         (*oFrame)(void);    // Ptr on the callback Frame routine
    void         (*oVSync)(void);    // Ptr on the callback VSync routine
    void         (*oDMAEc)(void);   // Ptr on the callback DMAEc routine
};

};
```

Call example in C:

```
int32_t      status;
const cnfIms2_t configure = {
    .oAcqMode      = KCONT;
    .oPixMode       = KCONTRAST10;
    .oAcqPage       = KEXTERNAL;
    .oNbRows        = 200;
    .oNbCols        = 200;
    .oKernSync      = (1<<BIMSEMA) | (1<<BVSEMA);
    .oHSync          = 0;
    .oFrame          = 0;
    .oVSync          = 0;
    .oDMAEc         = _myRoutine;
};

...
status = ims2_configure(&configure);

static void _myRoutine(void) {
...
}
```

int32_t ims2_acquisition(void)

Acquire an image into the memory (internal or external). This call launches all the necessary phases to integrate an image and to transfer it into the RAM.

input:	-		
output:	status	KIMS2NOERR	OK
		KIMS2TIMEO	Timeout error
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = ims2_acquisition();
```

```
int32_t ims2_getImage(void **image)
```

Get the image pointer.

input:	**image	Ptr to an image
output:	status	KIMS2NOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
int32_t      status;
uint8_t      *image;
...
status = ims2_getImage(&image);
```

```
int32_t ims2_ctrlLeds(uint8_t ledStates)
```

Control of the LEDs.

input:	ledState	State of the LEDs (0 = off, 1 = on)
output:	status	KIMS2NOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = ims2_ctrlLeds(0b00001010);
```

```
int32_t ims2_getRegister(S2Order_t *order)
```

Get the value of a S2 register.

input:	*order	Ptr on the order pack
output:	status	KIMS2NOERR OK
		KIMS2TIMEO Timeout error
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct S2Order {
    uint8_t      oSPICmd2          // SPI CMD P2
    uint32_t     oRegister;        // Register address
    uint8_t      oSPICmd4;
    bool_t       oRdDummy;
    uint32_t     oNbParam;         // Number of parameters
    uint32_t     oParam;           // Parameters
};
```

Call example in C:

```
int32_t      status;

// Read the S2 id order

S2Order_t    vOrder_RID = {
    SPI_CMD_READ_4_BYTE_P2,    // The command
    REG_S2_ID,                // The register address
    SPI_CMD_READ_4_BYTE_P4,    // P4 command
    TRUE,                     // Dummy read
    4,                        // 4 bytes
    -1                        // result
};

...
status = ims2_getRegister(&vOrder_RID);
iotx_printf(KSYST, "S2 id %8X\n", vOrder_RID.oParam);
```

Call example in C:

```
int32_t      status;
uint32_t     i, vTabLUTDay[1024];

// Read the LUT

S2Order_t    vOrder_RLU = {
    SPI_CMD_READ_4_BYTE_P2,    // The command
    0x60000,                  // The register address
    SPI_CMD_READ_4_BYTE_P4,    // P4 command
    TRUE,                     // Dummy read
    4,                        // 4 bytes
    -1                        // result
};

...
for (i = 0; i < 1024; i++) {
    vOrder_RLU.oRegister = 0x60000 + (i*4);
    if (ims2_getRegister(&vOrder_RLU) == FALSE) {
        return (KIMS2TIMEO);
    }
    aTabLUTDay[i] = vOrder_RLU.oParam;
}
```

```
int32_t ims2_putRegister(S2Order_t *order)
```

Put a value into a S2 register.

input:	*order	Ptr on the order pack
output:	status	KIMS2NOERR OK
		KIMS2TIMEO Timeout error
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Supported structure elements:

```
struct S2Order {
    uint8_t      oSPICmd2          // SPI CMD P2
    uint32_t     oRegister;        // Register address
    uint8_t      oSPICmd4;         // SPI CMD P4
    bool_t       oRdDummy;         // Dummy read
    uint32_t     oNbParam;         // Number of parameters
    uint32_t     oParam;           // Parameters
};
```

Call example in C:

```
int32_t      status;

// Write the H blanking order

S2Order_t    vOrder_HOR = {
    SPI_CMD_WRITE_2_BYTE,           // The command
    REG_DAU_HORIZ_BLANK,           // The register address
    -1,                            // P4 command
    FALSE,                          // Dummy read
    2,                             // 4 bytes
    -1                            // Value
};

...
vOrder_HOR.oParam = 512;
status = ims2_getRegister(&vOrder_HOR);
```

Call example in C:

```
int32_t      status;
uint32_t      i;
const uint32_t    aTabLUTDay[1024] = { 0x00025FFF, 0x000007FF, ... };

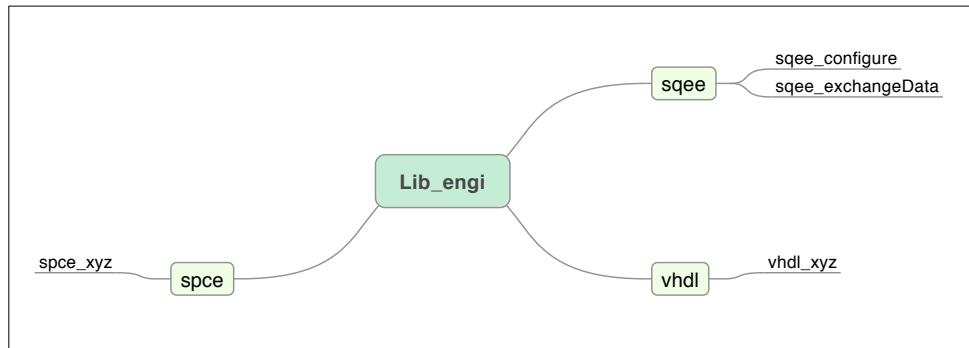
// Write the LUT

S2Order_t    vOrder_WLU = {
    SPI_CMD_WRITE_4_BYTE,           // The command
    0x60000,                      // The register address
    -1,                           // P4 command
    FALSE,                         // Dummy read
    4,                            // 4 bytes
    -1                            // Value
};

...
for (i = 0; i < 1024; i++) {
    vOrder_WLU.oRegister = 0x60000 + (i*4);
    vOrder_WLU.oParam    = aTabLUTDay[i];
    if (ims2_putRegister(&vOrder_WLU) == FALSE) {
        return (KIMS2TIMEO);
    }
}
```

3.7. The lib_engi library

This library is intended to control third party complex engines to extend the μKOS capability. The idea is to give the possibility to an μKOS process to exchange and to compute complex data via the capabilities of the supported engine.



Sysquake Embedded™ (<http://www.calerga.com>) is a splendid engine for such extensions.

sqee System Calls	
sqee_configure	Configure the sqee manager
sqee_exchangeData	Exchange data with the sqee manager

3.7.1. “sqee” Sysquake Embedded Engine

```
int32_t sqee_configure(cnfSqee_t *configure)
```



Configure the **sqee** manager.

```
input:      *configure          Ptr on the configuration buffer
output:     status              KSQEENOERR   OK
                      KSQEEGEERR Configuration error
reentrancy: YES
```

Supported structure elements:

```
struct cnfSqee_t {
    void        **oSqeeRef;           // Ptr on the Sqee (handle)
    uint32_t    oSize;               // Size of the memory
    uint8_t     *oMemory;             // Ptr on the memory location
};
```

Call example in C:

```
int32_t      status;
void         *sqeeHandle;
cnfSqee_t    configure;
...
configure.oSize = 90000;
configure.oMemory = (uint8_t *)syos_malloc(KINTERNAL, configure.oSize);
if (configure.oMemory == 0) {
    exit(EXIT_FAILURE);
}

// Configure the Sysquake Embedded (sqee) session

status = sqee_configure(&configure);

sqeeHandle = configure.oSqeeRef;
...
```

int32_t sqee_exchangeData(ioSqee_t *sqeData)

Exchange data with the **sqee** manager.

input:	*sqeData	Ptr on the data structure
output:	status	KSQEENOERR
reentrancy:	YES	OK

Supported structure elements:

```
struct cnfSqee_t {
    void        *oSqeeRef;           // Ptr on the Sqee
    uint8_t     oAction;            // Action
#define      KPUT          0           // Put to sqee
#define      KCOMPUTE       1           // Compute
#define      KGET          2           // Get from the sqee
#define      KCMDLINE       3           // Command line

    uint32_t    oNbDimensions;      // Dimensions (0=scalar, -1=null, 1..3=array)
#define      KSCALAR        0           // Scalar
#define      K1DARRAY       1           // 1D array
#define      K2DARRAY       2           // 2D array
#define      K3DARRAY       3           // 3D array

    uint32_t    oDimensions[3];     // Dimensions
    float64_t   *oPutData;          // Ptr on the data to put
    float64_t   *oGetData;          // Ptr on the data to get
    char_t      *oSqeeCommand;      // Ptr on the sqee command
};
```

Call example in C:

```
#define      KNBZS      3      // Number of z
#define      KNCOLS     5      // Number of cols
#define      KNROWS     4      // Number of rows

float64_t      result[KNBZS][KNROWS][KNCOLS];
cnfSqee_t      configure;
ioSqee_t       data;

...

// Configure the Sysquake Embedded (sqee) session

configure.oSize = 90000;
configure.oMemory = (uint8_t *)syos_malloc(KINTERNAL, configure.oSize);
if (configure.oMemory == 0) {
    exit(EXIT_FAILURE);
}
sqee_configure((cnfSqee_t *)&configure);

// Generate a random 3D array, rows = 5, lines = 4, z = 3

data.oAction      = KCOMPUTE;           // Compute ...
data.oSqeeCommand = "result = rand(4,5,3)"; // The command
sqee_exchangeData((ioSqee_t *)&data);   // Execution

// Ask for the 3D matrix

data.oAction      = KGET;               // Get ...
data.oNbDimensions = K3DARRAY;          // 3D array
data.oDimensions[0] = KNROWS;           // Rows
data.oDimensions[1] = KNCOLS;           // Cols
data.oDimensions[2] = KNBZS;            // z
data.oGetData     = &result[0][0][0];    // The 3D array result
data.oSqeeCommand = "xdata(result);";   // The command
sqee_exchangeData((ioSqee_t *)&data);   // Execution
...
```

3.8. References

Motorola, "Programmer's Reference Manual" M68000PM/AD Rev.1, 1992.

Motorola, "16/32-bit Application Manual" AD55, 1990.

Red Hat Software, Inc, "Official redhat 5.2 Linux OS" Red Hat, 1998.

Van Sickle Ted, "Programming Micro-controllers in C, 2nd ed." LLH, 2000.

3.9. Links

<http://www.gnu.org/>

<http://lists.gnu.org/>

<http://sourceware.org/newlib/>

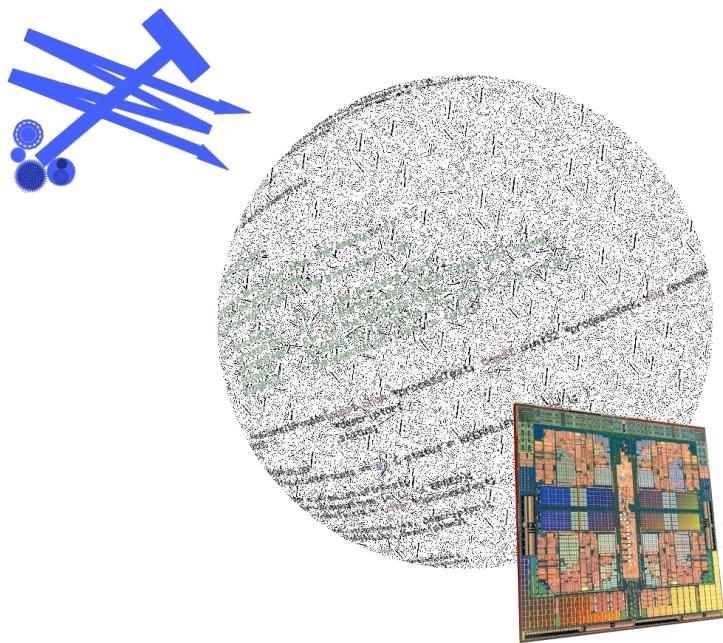
<http://www.embedded.com/>

<http://www.netrino.com/Publications/>

<http://www.billgatliff.com/>

<http://www.calerga.com/>

4. The μKernel



4.1. Introduction

Modern OS like Linux, MacOSX, UNIX, or high performance embedded systems like RTEMS, TRON, eCOS etc. are built around μKernels responsible for managing and optimizing the CPU time occupation for ensuring the best reactivity of a system. The basic idea behind the multitasking kernels is to give the possibility for a CPU to run quasi simultaneously (concurrently) many programs. In such a context, a program is periodically executed for a limited amount of time, only after that, it is unscheduled and another program is executed. This happens a thousand times every second, giving the impression to the system that all the processes are running at the same time. Of course, if a program needs to wait for particular conditions, it should return the control to the μKernel to avoid active waiting that would use the CPU resource for nothing. The multitasking programming approach is also very practical because it permits to decompose more clearly a given problem. However, some nasty situations like inter-process dead lock conditions, priority inversions, etc. can occur, making the debugging of such situations relatively tricky. Here is an example of code for an RS232 terminal application:

```
// In a classical implementation (1 loop)

main() {
    while (TRUE) {
        (if getReceiverStatus() == FULL)
            display(getRX_RS232());

        (if getTransmitterStatus() == EMPTY)
            (if getKeyboardStatus() == KEYPRESSED)
                putTX_RS232(getKeyboardData());

    }
}

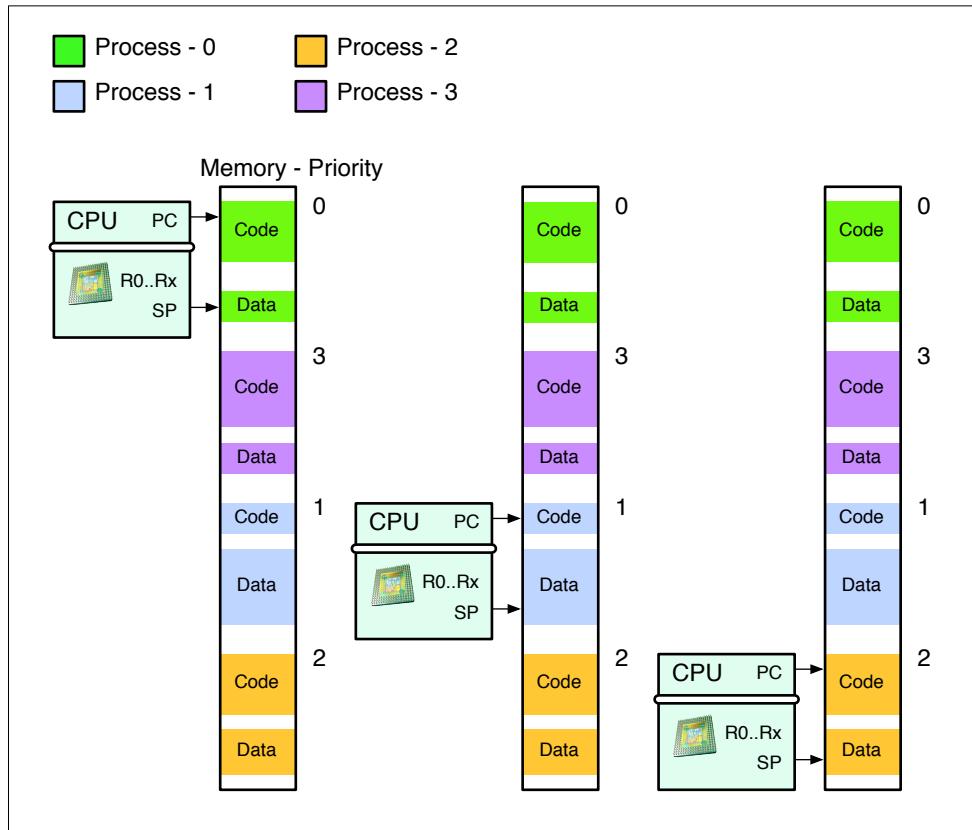
// In a multitasking implementation (2 independent processes)

process_receiver(const void *argument) {
    while (TRUE) {
        kern_waitSynchro(KSYNC_SC0R);      // Waiting for a char on the serial
        display(getRX_RS232());
    }
}

process_transmitter(const void *argument) {
    while (TRUE) {
        kern_waitSynchro(KSYNC_SC0T);      // Waiting for a free transmitter
        kern_waitSynchro(KSYNC_KEYB);      // waiting for a char on the keyboard
        putTX_RS232(getKeyboardData());
    }
}
```

4.2. Basic concepts

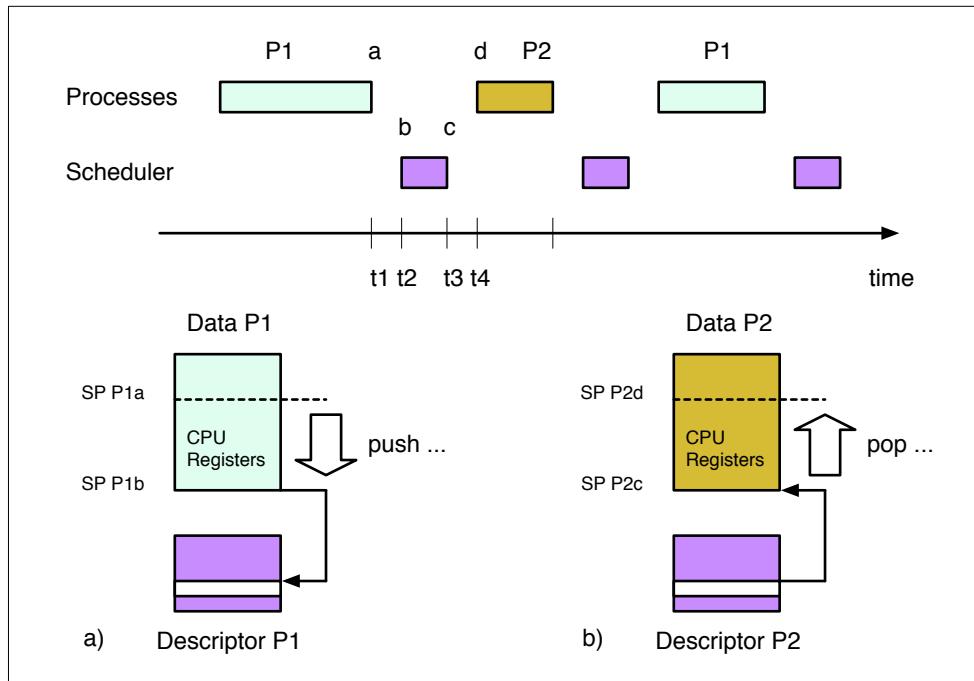
4.2.1. One CPU but many processes



The memory of the system contains the code and the data (data + stack) of all the created processes. When the μKernel decides to switch the context (to stop the execution of a process for a moment and to resume the execution for another one), it saves the current context on the process stack and it recovers from the stack of the new selected process all the information of the next context. At this time, a new process is executed from the state just before it was previously switched. The most important information for each process is stored in a data structure named **process descriptor**. All the process descriptors are stored in a particular memory area under the control of the μKernel.

The next picture illustrates the mechanism used by the μKernel for the context switching. In this example, 2 processes are scheduled and share the CPU bandwidth.

Picture "a" shows how the context of the process P1 is saved. Picture "b" shows how to restore the context of the process P2. Sequentially, here are the different steps:



1. At time **t1** the μKernel stops the execution of P1 and saves its context (all the CPU registers) on the stack of the process.
2. At time **t2** the μKernel saves the new stack **SP P1b** inside the process descriptor P1.
3. During **t2** and **t3** the scheduler of the μKernel determines which process has the highest priority (in this case P2).
4. At time **t3** the μKernel sets the stack pointer register of the CPU with the value of the stack (**SP P2c**) stored inside the process descriptor P2.
5. At time **t4** the context (all the CPU registers) is restored. The process P2 can now continue to execute its code from the state before it was previously switched.

In a more concrete way, here is the switching mechanism for the **Cortex M3-M4-M7** and for the **icyflex-1** cores:

Change of the context for Cortex M3 cores

```
/*
 * \brief Time slice timeout
 *
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          IOFF
 *
 *          ->      xPSR
 *          ->      PC
 *          ->      LR
 *          ->      R12
 *          ->      R3..R0      Block stacked by an exception
 *
 *          ->      R11..R4
 *          ->      BASEPRI
 *          ->      LR(R14)      Block stacked manually
 *
 * !!! Do not generate prologue/epilogue sequences
 *
 */

// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif
void  TIM2_IRQHandler(void) __attribute__ ((naked));
void  TIM2_IRQHandler(void) {

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid      i                  \n"      // 
        "mrs       r0,msp              \n"      // r0 = stack
    )
}
```

```
"mrs      r1,basepri      \n"    // r1 = basepri
"stmdb    r0!,{r1,r4-r11}  \n"    // Save the register list
"stmdb    r0!,{r14}        \n"    // Save the EXCEPTION return
"msr     msp,r0          \n"    //
"str     r0,%0          \n"    // Save the stack of the old process
"cpsie   i              \n"    //
:
: "m" (vKern_stackProcess)
: KSAVEREGISTERS
);

// Change the context due to a timeout

_newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
"cpsid   i              \n"    //
"ldr     r0,%0          \n"    // Recover the stack of the new process
"ldmia   r0!,{r14}        \n"    // Restore the EXCEPTION return
"ldmia   r0!,{r1,r4-r11}  \n"    // Restore the register list
"msr     msp,r0          \n"    // New stack
"msr     basepri,r1      \n"    // Restore the basepri
:
: "m" (vKern_stackProcess)
: KSAVEREGISTERS
);

__asm__ volatile (
"cpsie   i              \n"    //
"dmb
"dsb
"isb
"bx     lr              \n"    // Return
);
}
```

```
/*
 * \brief _newContextTOU
 *
 * - Change the context due to a timeout
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextTOU(void) {

    _timeStop();
    if (TIM2->SR & TIM_SR UIF) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackSlow();
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess, \
                  vTimeStart, \
                  vTimeStop, \
                  vTimeLastStart, \
                  vTimeException);
    vTimeException = 0;
#endif
}
```

Change of the context for Cortex M4 cores

```
/*
 * \brief Time slice timeout
 *
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          Normal stack frame      Stack frame with fpu
 *          IOFF                  IOFF
 *
 *          ->                   FPSCR
 *          ->                   S15..S0
 *          ->      xPSR        xPSR
 *          ->      PC          PC
 *          ->      LR(R14)    LR(R14)
 *          ->      R12         R12
 *          ->      R3..R0     R3..R0      Block stacked by an exception
 *
 *          ->      R11..R4    R11..R4
 *          ->      BASEPRI   BASEPRI
 *          ->                   S31..S16
 *          ->      LR(R14)    LR(R14)      Block stacked manually
 *
 *
 * !!! Do not generate prologue/epilogue sequences
 */
// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif
void  TIM2_IRQHandler(void) __attribute__ ((naked));
void  TIM2_IRQHandler(void) {

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid      i              \n"      // 
        "mrs       r0,psp           \n"      // r0 = stack
    )
}
```

```

"mrs      r1,basepri          \n"    // r1 = basepri
"stmdb    r0!,{r1,r4-r11}     \n"    // Save the register list
"tst     r14,#0x10           \n"    //
"it      eq                 \n"    //
"vstmdbeq r0!,{s16-s31}     \n"    // If used, save the fp registers
"stmdb    r0!,{r14}           \n"    // Save the EXCEPTION return
"msr     psp,r0             \n"    //
"str     r0,%0              \n"    // Save the stack of the old process
"cpsie   i                  \n"    //
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

// Change the context due to a timeout

_newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
"cpsid   i                  \n"    //
"ldr     r0,%0              \n"    // Recover the stack of the new process
"ldmia   r0!,{r14}           \n"    // Recover the EXCEPTION return
"tst     r14,#0x10           \n"    //
"it      eq                 \n"    //
"vldmiaeq r0!,{s16-s31}     \n"    // If used, restore the fp registers
"ldmia   r0!,{r1,r4-r11}     \n"    // Restore the register list
"msr     psp,r0             \n"    // New stack
"msr     basepri,r1         \n"    // Restore the basepri
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

__asm__ volatile (
"cpsie   i                  \n"    //
"dmb    \n"    //
"dsb    \n"    //
"isb    \n"    //
"bx    lr                \n"    // Return
);
}

```

```
/*
 * \brief _newContextTOU
 *
 * - Change the context due to a timeout
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextTOU(void) {

    _timeStop();
    if (TIM2->SR & TIM_SR UIF) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackSlow();
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess, \
                  vTimeStart, \
                  vTimeStop, \
                  vTimeLastStart, \
                  vTimeException);
    vTimeException = 0;
#endif
}
```

Change of the context for Cortex M7 cores

```
/*
 * \brief Time slice timeout
 *
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          Normal stack frame      Stack frame with fpu
 *          IOFF                  IOFF
 *
 *          ->                   FPSCR
 *          ->                   S15..S0
 *          ->      xPSR        xPSR
 *          ->      PC          PC
 *          ->      LR(R14)    LR(R14)
 *          ->      R12         R12
 *          ->      R3..R0     R3..R0      Block stacked by an exception
 *
 *          ->      R11..R4    R11..R4
 *          ->      BASEPRI   BASEPRI
 *          ->                   S31..S16
 *          ->      LR(R14)    LR(R14)      Block stacked manually
 *
 *
 * !!! Do not generate prologue/epilogue sequences
 */
// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif
void  TIM2_IRQHandler(void) __attribute__ ((naked));
void  TIM2_IRQHandler(void) {

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid      i              \n"      // 
        "mrs       r0,psp           \n"      // r0 = stack
    )
}
```

```

"mrs      r1,basepri          \n"    // r1 = basepri
"stmdb    r0!,{r1,r4-r11}     \n"    // Save the register list
"tst     r14,#0x10           \n"    //
"it      eq                 \n"    //
"vstmdbeq r0!,{s16-s31}     \n"    // If used, save the fp registers
"stmdb    r0!,{r14}           \n"    // Save the EXCEPTION return
"msr     psp,r0             \n"    //
"str     r0,%0              \n"    // Save the stack of the old process
"cpsie   i                  \n"    //
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

// Change the context due to a timeout

_newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
"cpsid   i                  \n"    //
"ldr     r0,%0              \n"    // Recover the stack of the new process
"ldmia   r0!,{r14}           \n"    // Recover the EXCEPTION return
"tst     r14,#0x10           \n"    //
"it      eq                 \n"    //
"vldmiaeq r0!,{s16-s31}     \n"    // If used, restore the fp registers
"ldmia   r0!,{r1,r4-r11}     \n"    // Restore the register list
"msr     psp,r0             \n"    // New stack
"msr     basepri,r1         \n"    // Restore the basepri
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

__asm__ volatile (
"cpsie   i                  \n"    //
"dmb    \n"    //
"dsb    \n"    //
"isb    \n"    //
"bx    lr                \n"    // Return
);
}

```

```
/*
 * \brief _newContextTOU
 *
 * - Change the context due to a timeout
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextTOU(void) {

    _timeStop();
    if (TIM2->SR & TIM_SR UIF) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackSlow();
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess, \
                  vTimeStart, \
                  vTimeStop, \
                  vTimeLastStart, \
                  vTimeException);
    vTimeException = 0;
#endif
}
```

Change of the context for icyflex-1 cores

```

;
; Time slice timeout
; /////////////////////
;
; - Save the context, Change the context, Recover another context.
;

handle_kernel_switcher:
    mmovl      -(px7),r0-r7          ; r7 .. r0
    mmovl      -(px7),px0-px6        ; px6 .. px0
    pmov       r3:r2,psu1           ; PB PA DM PF
    pmov       r1:r0,psu0           ; HD EC EX IN
    mmovl      -(px7),r0-r3          ;
    ;
    movil.h    r0.u,#0              ;
    pmov       r0.l,iel             ;
    movbp     +(px7,-4),r0          ; 0x00iel
    movil.h    r0.l,#0              ;
    pmov       iel,r0.l            ; ioff
    pmov       r0,sbr              ;
    movbp     +(px7,-4),r0          ; sbr
    ;
    movuiip   r0,#0                ;
    pmov       r0.l,scnt             ;
    addi      r0,#-1               ;
    jmp       pa2,lf               ;
    nop                   ;
    ;
    loop      r0.l,1f               ;
    pop       r3:r2                ; 64-bits HS
    movbp     +(px7,-4),r2          ; LSB
    movbp     +(px7,-4),r3          ; MSB
1:   addil    r0,#1                ;
    movbp     +(px7,-4),r0          ; scnt
    ;
    movil.h    px0.u,#sval.u(_vStckProcess)  ;
    movil.h    px0.l,#sval.l(_vStckProcess)  ;
    movbl     (px0,0),px7          ; Save the ptr on the context

; Call the nano kernel for switching a task

    sjsr      _kernel_switcher          ; Call the nano kernel
    nop                   ;

```

```

; Restore the new context

    movil.h      px0.u,#sval.u(_vStckProcess)      ;
    movil.h      px0.l,#sval.l(_vStckProcess)      ;

    movbl       px7,(px0,0)                         ; Recover the ptr on the context
    movbp       r0,(px7,4)+                          ; scnt
    addil       r0,#-1                             ;
    jmp        pa2,1f                             ;
    nop                                     ;

    loop        r0.1,1f                           ;
    movbp       r3,(px7,4)+                         ; MSB
    movbp       r2,(px7,4)+                         ; LSB
    push        r3:r2                            ; 64-bits HS

1:   movbp       r0,(px7,4)+                         ; sbr
    pmov        sbr,r0                           ;
    movbp       r0,(px7,4)+                         ; 0x00iel
    pmov        iel,r0.1                         ;

    mmovl       r0-r3,(px7)+                        ;
    pmov        psu0,r1:r0                         ; HD EC EX IN
    pmov        psu1,r3:r2                         ; PB PA DM PF
    mmovl       px0-px6,(px7)+                      ; px6 .. px0
    mmovl       r0-r7,(px7)+                        ; r7 .. r0
    rte          r0                               ;
    nop                                     ;

/*
 * Time slice timeout
 * /////////////
 *
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 *
 */
void  kernel_switcher(void) {

// INT acknowledge and new time for the next process
// Change the context and prepare the next process

    *TIMER1_STATUS = (1<<BBOUND_REACHED);
    *TIMER1_CTRL = 0;
    scheCallBackSlow();
}

```

4.2.2. Temporal aspects

Each process can only use the CPU for a limited amount of time (defined by the time slice). After this time (process timeout), the μKernel stops its execution and tries to figure-out the next highest priority available process (the choice is made by the **scheduler**). In case that there are more processes having the same priority, the scheduler selects the process with the highest priority that has been waiting on the CPU for the longest time. The time slice has to be defined short enough to permit a good reactivity for all the installed processes, and long enough to avoid large overheads due to the multiple context switching.

4.2.3. Reentrancy

If a piece of code can be executed by more than a process at the same time, it is mandatory to avoid using static variables. Static variables can be used only with an **exclusion semaphore** or encapsulated by the macros **INTERRUPTION_OFF** and **INTERRUPTION_RESTORED** during all the life of the static variable.

Interruption control for Cortex M3-M4-M7 cores

```
#define INTERRUPTION_OFF volatile uint32_t __saveBASEPRI; \
    __asm__ volatile(" \
        mrs r0,basepri \
        str r0,%0 \
        mov r0,%1 \
        msr basepri,r0" \
        : \
        : "m" (__saveBASEPRI), \
        "i" (KINT_IMASK_OFF<<4) \
        : "r0" \
    ); \
 \
#define INTERRUPTION_RESTORED __asm__ volatile (" \
    ldr r0,%0 \
    msr basepri,r0" \
    : \
    : "m" (__saveBASEPRI) \
    : "r0" \
);
```

Interruption control for icyflex-1 cores

```
#define INTERRUPTION_OFF volatile uint16_t __saveIEL; \
    __asm__ (" \
        pmov      r7.l,iel \
        movbl.h  %0,r7.l \
        movil.h  r7.l,#%1 \
        pmov      iel,r7.l" \
    : \
    : "m"  (__saveIEL), \
        "i"  (KINT_IMASK_PERIPH) \
    : "r7" \
    );

#define INTERRUPTION_RESTORED __asm__ (" \
    movbl.h  r7.l,%0 \
    pmov      iel,r7.l" \
    : \
    : "m"  (__saveIEL) \
    : "r7" \
    );
```

4.2.4. Preemption

When necessary, the μKernel can stop the execution of a process and run a more priority one. This occurs naturally after the process timeout, or when hardware interruptions communicate **synchro events** to the μKernel.

4.2.5. Inversion of priority

Priority inversion is the situation where a low priority process holds a high priority process (i.e. suspended by a semaphore). This causes the execution of the high priority process to be blocked until the low priority process has released the resource (i.e., in a semaphore), effectively **inverting** the **relative priorities** of the two processes.

In some cases, priority inversion can occur without causing immediate harm; the delayed execution of the high priority process goes undetected. Priority inversion can also reduce the perceived performance of the system. Low priority processes usually have a low priority because it is not important for them to finish promptly. On the other hand, a high priority process has a high priority because it is more likely to be subjected to strict time constraints. Because priority inversion results in the execution of the low priority process blocking the high priority process, it can lead to reduced system responsiveness, or even the violation of response time guarantees.

4.2.6. Critical resources

As for the static variables, the access to the peripherals and shared memory structures has to be done via an **exclusion semaphore** or encapsulated by the macros **INTERRUPTION_OFF** and **INTERRUPTION_RESTORED** (useable only for processes running in supervisor space). If the duration necessary to access the critical resource is short, the usage of the macros is the fastest and the most convenient way for the protection. In all the other cases, the usage of an **exclusion semaphore** is mandatory.

```
// Example of a simple protection of a critical resource

#define      KSZBUF      10

static uint8_t      vBuffer[KSZBUF];      // Critical resource

process_1(const void *argument) {
    uint8_t      i, result;

    while (TRUE) {
        for (i = 0; i < KSZBUF; i++) {
            INTERRUPTION_OFF;
            vBuffer[i] = 2 * i; result = summBuff(vBuffer);
            ...
            INTERRUPTION_RESTORED;
        }
    }
}

process_2(const void *argument) {
    uint8_t      i, result;

    while (TRUE) {
        for (i = 0; i < KSZBUF; i++) {
            INTERRUPTION_OFF;
            buffer[i] = 3 * i; result = summBuff(vBuffer);
            ...
            INTERRUPTION_RESTORED;
        }
    }
}

uint8_t      summBuff(uint8_t *buffer) {
    uint8_t      i, result = 0;

    for (i = 0; i < KSZBUF; i++) { result += *buffer++; }
    return (result);
}
```

4.3. The μKOS μKernel implementation

The μKOS μKernel implements all the basic concepts available in modern μKernels. The implementation of all the primitives is fully static, with in mind the idea to use this μKernel in ASICs, where resources (specially the memory) are often critical and limited. All the parameters of this μKernel can be tailored in order to obtain the best matching between the resources, the performance and the application.

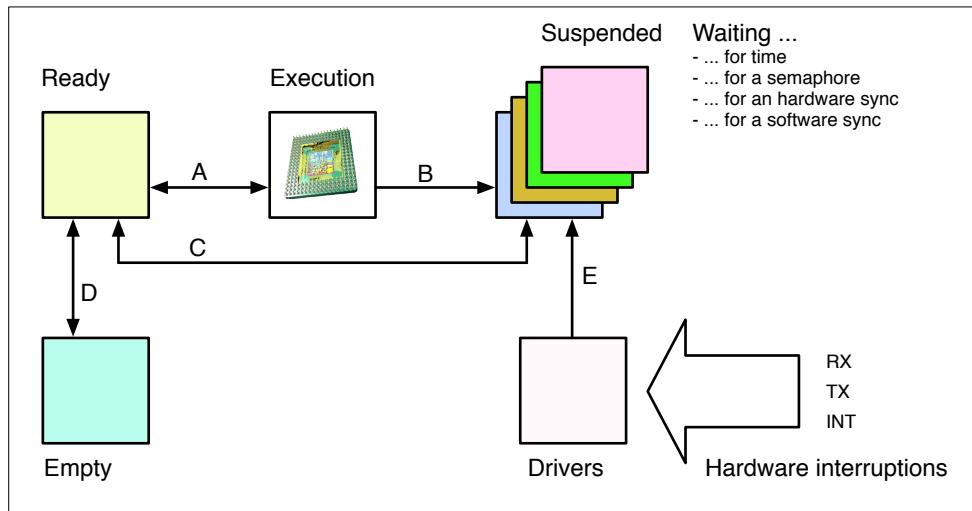
Basic features of the μKOS μKernel

μKOS μKernel implements the following basic features:

- Up to 255 static processes or overlays.
- Up to 255 dynamic priorities.
- Preemptive multitasking.
- Each process can run in supervisor or user spaces.
- Time management:
 - - Process timeout.
 - - Process suspension.
 - - Action waiting under timeout.
 - - Precise temporal execution of routines (for state machines).
- Up to 255 static semaphores.
- Up to 255 interprocess associations.
- Up to 255 interprocess mailboxes / queues.
- Up to 255 software timers.
- Up to 255 memory pools.
- Process synchronization (events).
- Process synchronization (signals).
- CPU usage statistics.
- Support of the UNIX time.
- Portable to any CPUs via stubs.
- ROMable.

4.3.1. State of a process

Each process can be in 4 precise states. The algorithm of the **scheduler** is responsible for selecting the right process that is ready to use the CPU (**A**). The **system calls** to the μKernel manage the state of the process accordingly to the feature that has the control of the process (semaphore, synchronization, time, etc.) (**B..E**).

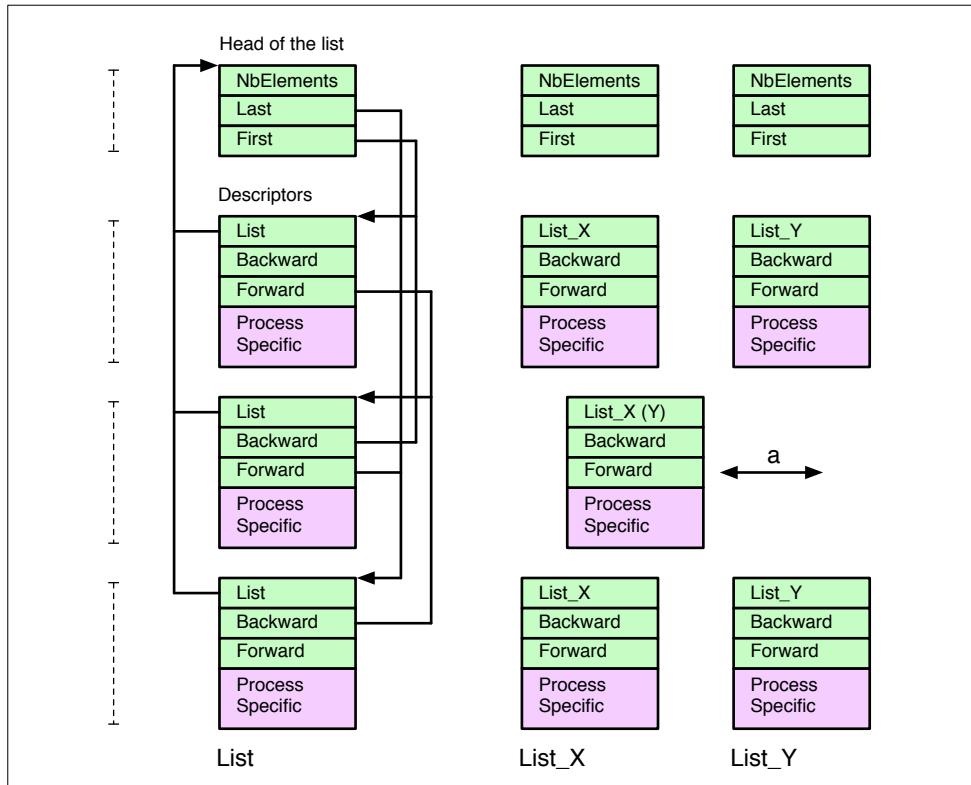


- **The empty state:** all the free (usable) processes.
- **The ready state:** all the processes that can use the CPU at the next context switching are in this state.
- **The execution state:** only the process using the CPU can be in this state.
- **The suspended state:** all the processes waiting for a particular event are in this state.

External hardware interruptions have an indirect effect on the μKernel. In order to obtain the best reactivity of the system, the interruption routines (out of the μKernel) should handle part (or the totality) of the interruption process. The user can decide to communicate (synchronize) the interruption events to the kernel via a **synchro event**. All the suspended processes waiting on such synchro events will be then placed in the **ready state**.

4.3.2. Process descriptors and lists

The **process descriptors** are a particular memory structure handling the information concerning the process, its state and its environment. These process descriptors are chained in **list** representing the different available states for a process (**Empty**, **Ready**, **Suspended**, etc.).



The list implements a backward/forward connection scheme. This double chaining scheme allows a fast connection/de-connection (**a**) of the process descriptors to/from a specific list (i.e **List_X** and **List_Y**). The list management is one of the key of the μKOS μKernel.

As previously introduced the process descriptor contains the important information about the process and its temporal and logic evolution. Here is the data structures used in μKOS:

```

struct proc {
    obje_t      oObject;           // Process object (connectable)
    spec_t      oSpecification;   // Process specification
    work_t      oInternal;        // Process internal stuff
    stts_t      oStatistics;      // Kernel statistic
    sync_t      oSynchro;         // Kernel synchro (evnt & sema)
};

struct list {
    proc_t      *oFirst;          // Ptr on the first process
    proc_t      *oLast;           // Ptr on the last process
    uint16_t    oNbElements;     // # of elements of the list
};

struct spec {
    const char_t *oIdentifier;   // Process Id
    const char_t *oText;         // Ptr on the process Id text

    volatile uint32_t *oStkSupBegin; // Ptr on the begin of the stack
    volatile uint32_t *oStkSup;     // Process stack (supervisor)
    volatile uint32_t *oStkUsr;     // Process stack (user)
    volatile uint32_t oSzStk$;      // Size of the process stack
    void (*oCode)\n(
        const void *arg); // Process code
    uint8_t oStackMode;           // Stack mode
    #define KSTKSTATIC 0 // KSTKSTATIC = static stack
    #define KSTKDYNAMIC 1 // KSTKDYNAMIC = dynamic stack

    uint8_t oKind;               // Process kind
    #define KPROCNORMAL 0 // KPROCNORMAL = normal
    #define KPROCDAEMON 1 // KPROCDAEMON = daemon
    #define KPROCOVERLAY 2 // KPROCOVERLAY = overlay

    uint8_t oMode;               // Process run mode
    #define KUSER 0 // KUSER = in a user
    #define KSUPERVISOR 1 // KSUPERVISOR = in a supervisor

    uint8_t oPriority;           // Process priority
    uint32_t oCommManager;       // Default I/O channel
    uint32_t oIdPack;            // Overlay pack of processes
};

struct sync {
    uint32_t oFlags;             // For cmsis compatibility
    uint32_t oListEvents;        // List of events
    uint32_t oListEventsPending; // List of pending events
    uint32_t oListSignals;       // List of signals
    uint32_t oListSignalsPending; // List of missed signals
    uint32_t oNbSyncSema;        // Number of a for a semaphore
};

```

```

struct work {
    uint8_t      oState;           // Process state
    #define BPROCINSTALLED 0        // Process installed
    #define BPROCRUNNING 1          // Process running
    #define BPROCSUSPTIME 2         // Process susp for a time
    #define BPROCSUSPEVNT 3         // Process susp for an event
    #define BPROCSUSPSIGN 4         // Process susp for a signal
    #define BPROCSUSPSEMA 5         // Process susp for a semaphore
    #define BPROCSUSPDEBG 6         // Process suspended for a debug

    int32_t       oStatus;          // Status
    uint32_t      oTimeout;         // A timeout
    uint32_t      oSkip;            // Number of skipped time
    uint8_t       oDynaPriority;    // Process priority dynamic

    const void    *oLocal;          // General Ptr
};

struct stts {
    uint64_t      oNbExecutions;    // # of time the process was scheduled
    uint64_t      oNbKernCalls;     // # of time that the kernel was called

    uint32_t      oAvStkSup;        // Available stack size
    uint16_t      oTimePMin;        // Min CPU time used by the Process
    uint16_t      oTimePMax;        // Max CPU time used by the Process
    uint16_t      oTimePAvg;        // Avg CPU time used by the Process
    uint64_t      oTimePCum;        // Cum CPU time used by the Process
    uint16_t      oTimeKMin;        // Min CPU time used by the kernel
    uint16_t      oTimeKMax;        // Max CPU time used by the kernel
    uint16_t      oTimeKAvg;        // Avg CPU time used by the kernel
    uint64_t      oTimeECum;        // Cum CPU time used by the kernel
    uint16_t      oTimeEMin;        // Min CPU time used by the Exceptions
    uint16_t      oTimeEMax;        // Max CPU time used by the Exceptions
    uint16_t      oTimeEAvg;        // Avg CPU time used by the Exceptions
    uint64_t      oTimeECCum;       // Cum CPU time used by the Exceptions
};

};

```

The main structure **proc_t** is composed of 5 different sections. The first section contains the elements of the **list_t** and the **proc_t** structures used to link the process to a specific list. Four more sections contain additional information necessary for the uKernel. The **spec_t** section contains all the information that the user has to provide, such as the stack, the execution address, the priority, etc. The **work_t** section contains the internal process status and evolution information; this is used only by the uKernel. The **stts_t** section contains all the statistic information related to the execution of the process, such as the uKernel overhead, the number of time that the process was scheduled, etc. The **evnt_t** section contains the information used for the event and for the synchronization logic.

4.3.3. Daemons

Daemons are processes that operate in background for supervising some parts of the system or the µKernel. In the µKOS implementation, the µKernel needs at least four daemons: the **idle** the **timeout** the **stack sanity** and the **software timers**.

The **idle** process is a special process having a very low priority. Normally it is scheduled to run only when all the available processes of the system are not ready in a suspended list. In this case, the scheduler will run this idle process until another process can be operational.

The **timeout** process has a very low priority. It scans periodically all the lists containing process in a suspended state. If a timeout is detected for one or more processes, then the µKernel re-places the detected process in the ready list returning an error.

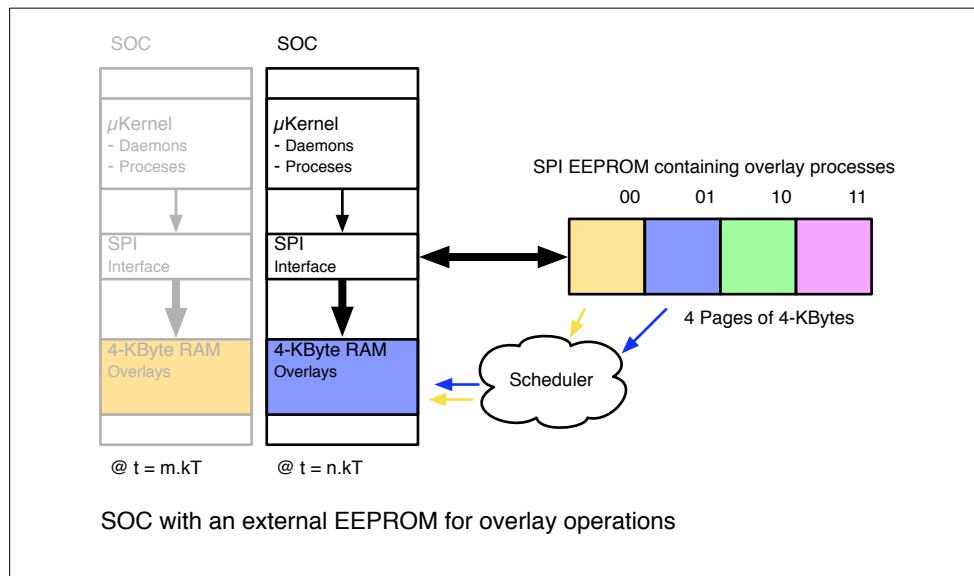
The **stack sanity** process has a very low priority. It scans periodically all the process stacks in order to verify its depth. The information concerning the available amount of place is available in the process descriptor .

The **software timer** process is responsible to manage the logic of all the software timers.

4.3.4. Overlay processes

In some CPU systems, and mainly for a cost reason, the size of the memory (**ROM** and **RAM**) has to be minimized. This is a particular and sensitive constraint in **SOCs** (System On Chips), where the surface of the silicon is proportional to the size of the memories integrated, and at the end, the surface of the silicon drives the final cost of the SOCs.

In many cases, it could be interesting to use an external **EPROM** or **SDRAM** containing the pieces of code that have to be executed. The uKernel responsible for managing the execution of the application, when necessary, loads the external code inside the RAM of the SOC for its execution; this mechanism is called **overlay**.



The usage of overlay processes can reduce the size of the memory for a system; on the other hand, the reduction of the memory size is paid with an increased μKernel overhead, necessary for reloading the internal SOC memory from an external page of code. The usage of 2 memories can reduce (and in some cases eliminate) the overhead; during the execution of one process inside the overlay **RAM page 1**, the overlay **RAM page 0** can be reloaded with the future process that should be executed as soon as the scheduler will take the decision.

The performance of a system dealing with overlay processes can be improved using specific hardware (i.e. an **SPI** coupled with a **DMA**) working in combination of many overlay **RAMs**. The scheduler logic has to be tailored accordingly to the selected architecture.

The customization of the μKernel is done via a **stub**, where the hardware as well as the kind of CPU is taken in account.

Here is an example of the **stub overlay** customization:

```
/*
 * kernel_loadOverlay
 * -----
 *
 * - Load an overlay process inside the overlay RAM.
 *
 */
int32_t      kernel_loadOverlay(uint32_t idModule) {
    bigPtrTable_t      *big;
    moduleHeader_t      *moduleCODE;
    uint16_t            i;
    uint32_t            *overlaySrc, *overlayDst, overlayLen;

    GETBIGPTR(big)
    moduleCODE = *big->oFrstModuleM;

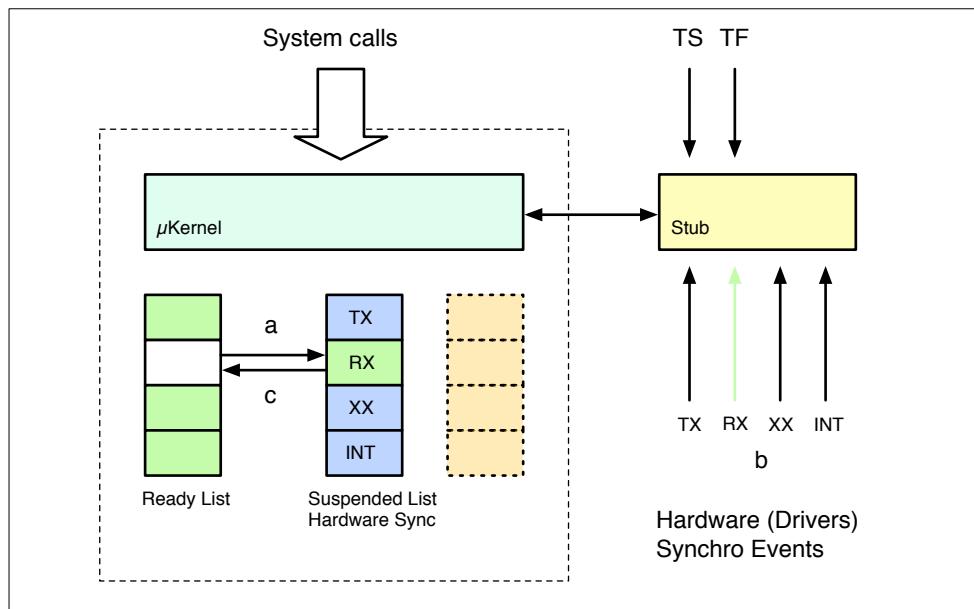
    do {
        if (moduleCODE->oIdentifier == idModule) {
            overlaySrc = (uint32_t *)moduleCODE;
            overlayDst = (uint32_t *)&vOverlay[0];
            overlayLen = moduleCODE->oLnEPROM;
            for (i = 0; i < (overlayLen/4)+1; i++) {
                *overlayDst++ = *overlaySrc++;
            }
        }

        return (KKERNNOERR);
    }
    moduleCODE = (MODULEHEADER *)((uint32_t)moduleCODE + \
        (uint32_t)moduleCODE->oLnEPROM);

    } while (moduleCODE->oIdentifier != KIDTERMINE);
    return (KKERNOVRLY);
}
```

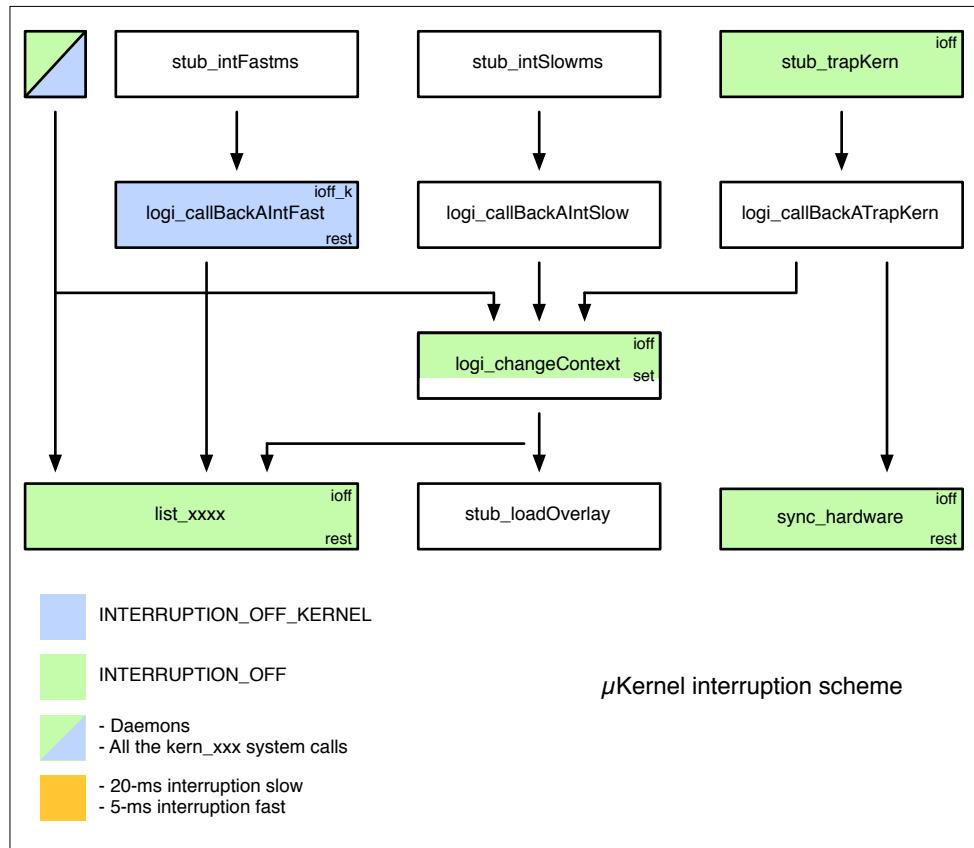
4.3.5. Interruptions

For its operations the μKernel requires a software interruption (SWI) as well as two hardware timer interruptions. The software interruption is needed for passing messages from the managers or from certain functions of the μKernel to the scheduler. The timer **TS** is used to generate the process timeout (time slice). In μKOS implementation this time is set to 20-ms for slow CPUs. The timer **TF** is used to manage the precise temporal aspects of the μKernel such as for the suspended processes, and the time is set to 1 to 5-ms (depending on the CPU performances). An additional free running timer is used for maintaining statistic information such as the CPU time used by a process and by the μKernel. This timer does not use the interruptions.



All the other interruptions are not directly managed by the μKernel. The idea is to handle the processing of the interruption inside the managers. This ensures the optimal reactivity of the system for all the temporal events. When necessary, the managers can trigger the μKernel via a **semaphore**. For example, a process may need to be synchronized each time a character is received by the RS232 manager. To do this, the process can be placed in the **suspended list (a)** until a **semaphore** (i.e. RS232 RX) is detected (**b**). When this event occurs it places the process inside the **ready list (c)**.

For maintaining a good reactivity of the system, the control of the critical resource accesses uses 2 interruption levels. The **INTERRUPTION_OFF** masks all the interruptions and has to be used only in limited regions of the code. The **INTERRUPTION_OFF_KERNEL** masks only the timers used by the μKernel. When this interruption level is used, it does not disturb the high priority interruptions coming from the peripherals. This ensures to have a correct protection of the μKernel features, and it maintains at the same time the necessary reactivity for serving the peripherals.

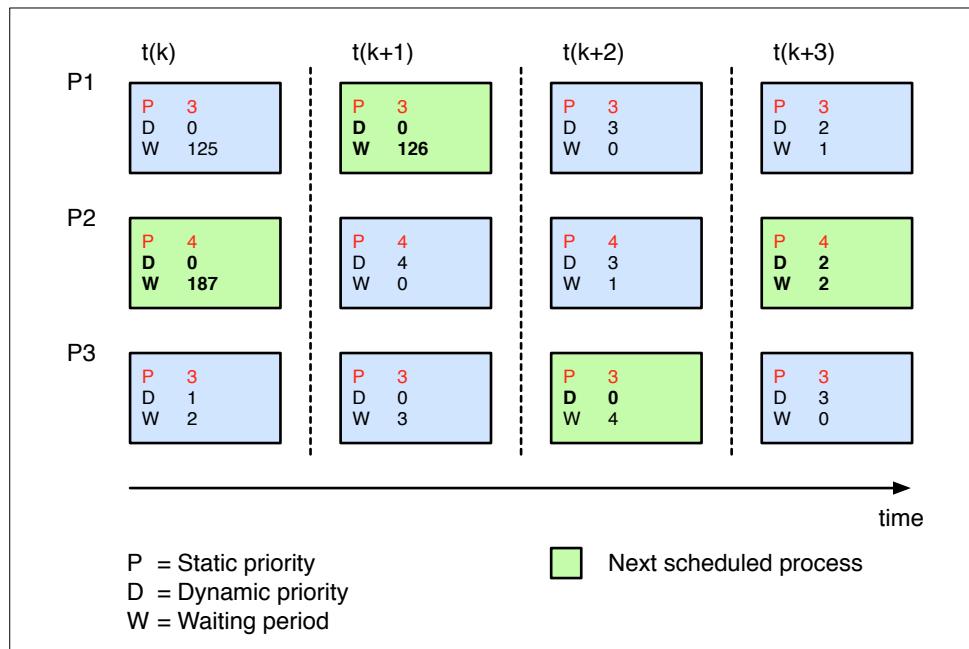


In the μKOS implementation, the system disables all the interruptions only inside the scheduler (see the routines **sche_changeContext** and **_getPriority**, and inside the routines responsible for connecting/de-connecting the process to/from the appropriate lists).

4.3.6. The scheduler

The scheduler is responsible for determining which process will obtain the CPU after the next context switching. The implemented logic is:

- Check for the process having the highest dynamic priority (0 is the highest priority).
- In case of many processes having the same priority, the one that has been waiting for the longest time is selected.
- All the other processes will then increase their priority (decrementing the dynamic priority register).
- The dynamic priority of the selected process is then set to its static value.

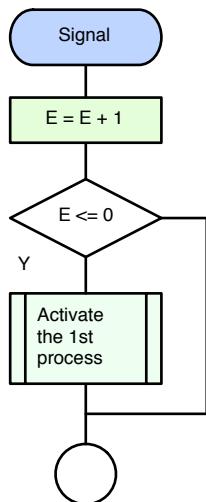
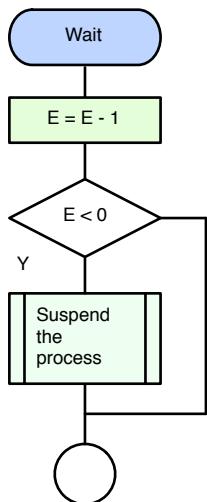


Here is an example with 3 processes (P1, P2 and P3). The bold parameters in the picture indicate the elements used by the scheduler for selecting the process that will run. The **$t(k+x)$** represents the process timeout and the moment where the scheduler has to decide which process will be running the CPU the next time; in this picture the selected process is the green one.

4.3.7. Semaphores

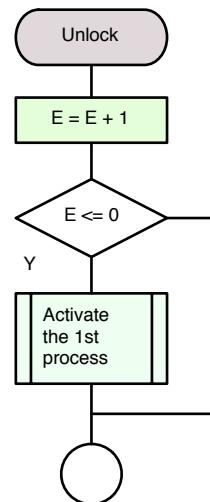
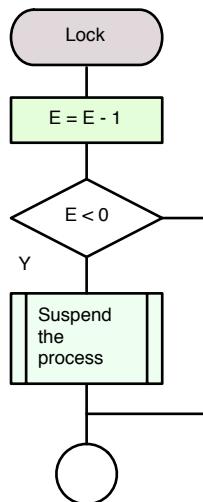
Synchronization semaphore

Initial condition of E = 0



Mutual Exclusion semaphore

Initial condition of E = 1



A semaphore is a generic mechanism for protecting particular critical resources or for synchronizing processes. The algorithm implemented uses a counter **E**. The initial value of **E** defines if the semaphore operates as a **synchro** (synchronization **E** = 0) semaphore or as a **mutex** (mutual exclusion **E** = 1) semaphore.

```

// Synchronization and mutual exclusion semaphore primitives

kern_createSyncSemaphore("sema", &handle) {
    static uint32_t E = 0;
    ...
}

kern_createMutexSemaphore("sema", &handle) {
    static uint32_t E = 1;
    ...
}
  
```

```
kern_waitSemaphore(...){  
kern_lockSemaphore(...){  
  
    E = E - 1;  
    if (E < 0) suspendProcess(...);  
}  
  
kern_signalSemaphore(...){  
kern_unlockSemaphore(...){  
  
    E = E + 1;  
    if (E >= 0) scheduleProcess(...);  
}
```

The semaphore stores the events (the E counter handles this information). This ensures a good synchronization mechanism.

Example of a synchronization semaphore; the RS232 manager uses a semaphore to **signal** that a character was received and is available. The process P1 uses this semaphore to be **synchronized** on this event.

```
// Example of a synchronization semaphore (interruption synchronize P1)  
  
static sema_t *vSemaphore;  
  
kern_createSyncSemaphore("int synchro", &vSemaphore);  
...  
  
interruption_RS232() {  
    ...  
    if (*QSM_SCSR & (1<<RDRF))  
        kern_signalSemaphore(vSemaphore); // Signal (generate a synchro)  
    ...  
}  
  
process_1(const void *argument) {  
    ...  
  
    while (TRUE) {  
        kern_waitSemaphore(vSemaphore, 1, -1); // Suspend P1  
                                         // waiting for the manager sig  
                                         // without timeout  
        display(getRX_RS232());  
                                         // Read and display the data  
    }  
}
```

Example of a mutual exclusion semaphore: a global buffer (critical resources) can be accessed by the processes P1, P2 and P3. The routine **computeBuffer** can be executed only by one process at the same time. The first process that calls the **lock** function (**kern_waitSemaphore**) has the resource until it calls the **unlock** function (**kern_signalSemaphore**). For this example it is supposed that the order of the access to the critical resource is: P3, P2 and P1. In the semaphore implementation only 2 system calls are available: the **kern_waitSemaphore** and the **kern_signalSemaphore**. For clarity reasons, 2 new system calls are aliases: **kern_lockSemaphore = kern_waitSemaphore** and **kern_unlockSemaphore = kern_signalSemaphore**.

```
// Example of a mutual exclusion semaphore

#define      KSZBUF      10

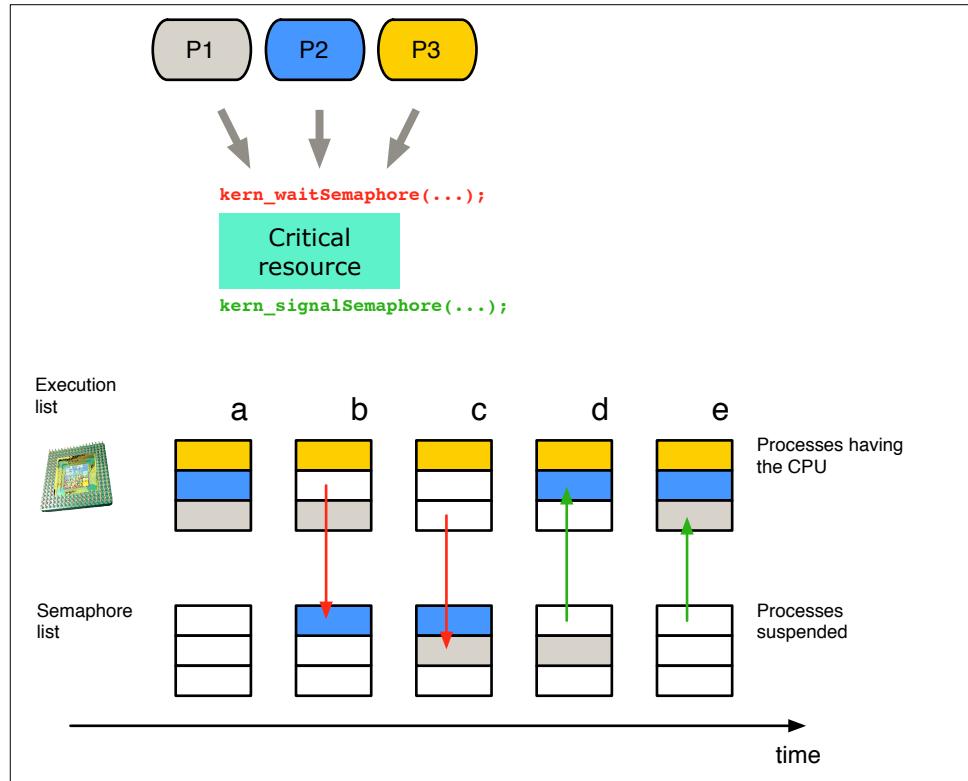
static uint32_t      vBuffer[KSZBUF];
static sema_t        *vSemaphore;

kern_createMutexSemaphore("buffer protection", &vSemaphore);
...

process_1(const void *argument) {
    ...
    kern_lockSemaphore(vSemaphore, -1);           // Lock the resource (no timeout)
    computeBuffer(&vBuffer[0]);                  // Buffer reserved for process_1
    kern_unlockSemaphore(vSemaphore);             // Unlock the resource
    ...
}

process_2(const void *argument) {
    ...
    kern_lockSemaphore(vSemaphore, -1);           // Lock the resource (no timeout)
    computeBuffer(&vBuffer[0]);                  // Buffer reserved for process_2
    kern_unlockSemaphore(vSemaphore);             // Unlock the resource
    ...
}

process_3(const void *argument) {
    ...
    kern_lockSemaphore(vSemaphore, -1);           // Lock the resource (no timeout)
    computeBuffer(&vBuffer[0]);                  // Buffer reserved for process_3
    kern_unlockSemaphore(vSemaphore);             // Unlock the resource
    ...
}
```



Here is the temporal sequence of all the lock-unlock calls by the processes that request the critical resource via a mutual exclusion semaphore.

- The process **P3 locks** the semaphore; it has the resources.
- The process **P2 tries** to lock the semaphore; the semaphore is already **locked by P3**, so, **P2 is unscheduled** and placed in the waiting list of the semaphore.
- The process **P1 tries** to lock the semaphore; the semaphore is already **locked by P3**, so, **P1 is unscheduled** and placed in the waiting list of the semaphore.
- The process **P3 has terminated** with the critical resource; it unlocks the semaphore; the semaphore is now **locked by P2**, so, **P2 is re-scheduled** and placed in the execution list.
- The process **P2 has terminated** with the critical resource; it unlocks the semaphore; the semaphore is now **locked by P1**, so, **P1 is re-scheduled** and placed in the execution list.

4.3.8. Associations - Shared Memory

The association primitives allow to **associate** a generic pointer to an **identifier**, and then to **publish** it. This simple mechanism allows to share data from a process, to one or many others using a simple high level reference.

For example, the process P1 is responsible for collecting the data coming from a sensor. It can process this data and publish the result to all the processes interested in this information.

```
// Example of an association published by P1 and used by P2

static uint32_t      vSensBuffer[KSZBUF];

process_1(const void *argument) {
    astp_t      *association;
    shar_t      pack;

    pack->oGeneral = &vSensBuffer[0];
    pack->oSize = KSZBUF * sizeof(uint32_t);

    kern_createAssociation("Sensor", &association);           // Create an association
    kern_publishAssociation(association, &pack);               // Publish association

    while (TRUE) {
        kern_lockSemaphore(...);                                // Lock the resource
        processSensor(vSensBuffer);                            // Process the sensor
        kern_unlockSemaphore(...);                             // Unlock the resource
    }
}

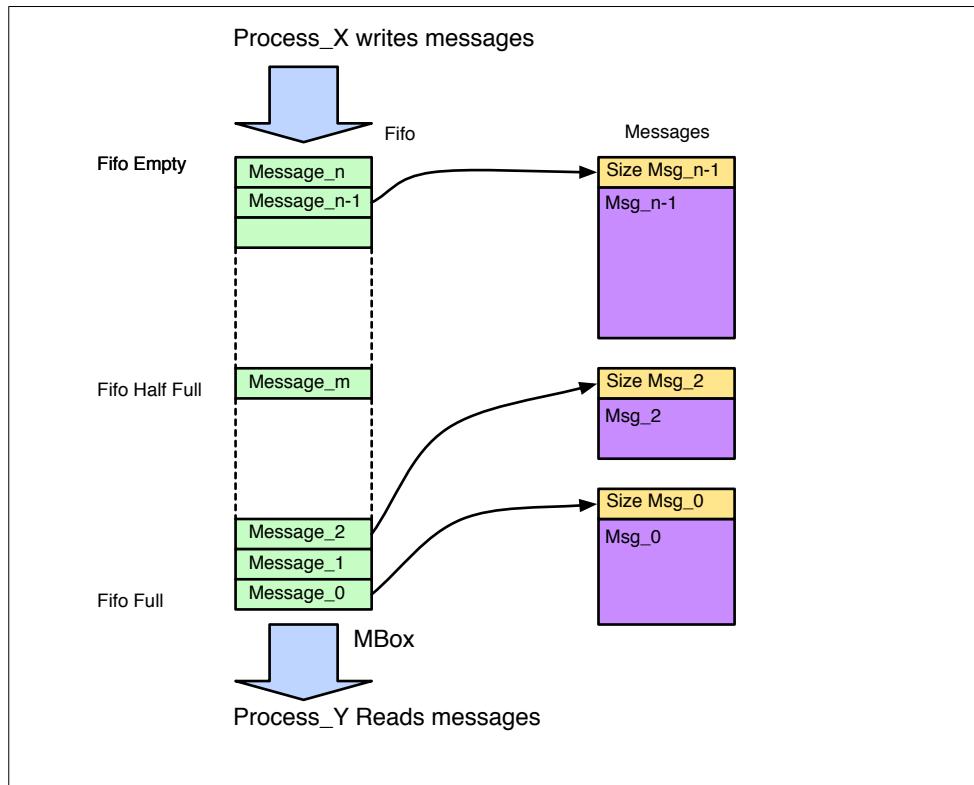
process_2(const void *argument) {
    astp_t      *association;
    shar_t      *pack;
    uint32_t     *ptr;

    kern_getAssociationById("Sensor", &association);
    kern_findAssociation(association, pack);                // Look for the "Sensor"

    ptr = (uint32_t *)pack->oGeneral;
    while (TRUE) {
        kern_lockSemaphore(...);                                // Lock the resource
        useSensorInformation(ptr);                           // Use the "Sensor" info
        kern_unlockSemaphore(...);                            // Unlock the resource
    }
}
```

4.3.9. Mailboxes & queues

Inter-process communications can be handled by **mailboxes** or by **queues**. Mailboxes are **fifo** (first-in first-out) structures containing message pointers. The default depth of the fifo is set to contain 32 messages. Each message pointer points to the memory areas containing messages of any kind of nature and size.



The fifo message structure contains not only the pointer to the message, but also the size of the message. The queues very similar to the mailboxes; the difference is that the message is handle only by a 32-bit word,

```
// Example of a mailbox (P1 sends a buffer, P2 receives it)

static void process_1(const void *argument) {
    uint8_t          *bufSnd;
    uint16_t         sizeSnd;
    uint32_t         counterSnd = 0;
    int32_t          status;
    volatile mbox_t  *mbox;

    status = kern_createMailbox("Mailbox 0 to 1", &mbox);
    if (status != KKERNNOERR) iotx_printf(KURTO, "No Mailbox 0 to 1\n");
    while (TRUE) {

// Send the message

        bufSnd = (uint8_t *)syos_malloc(sizeof(uint32_t)*1);
        if (bufSnd == 0) {
            iotx_printf(KURTO, "Error malloc 0\n"); terminate_process();
        }

        *bufSnd = counterSnd++;
        do {
            kern_switchFast();
        } while (kern_writeMailbox(mailBox, bufSnd, sizeSnd) == KKERNTOPAC);
    }
}

static void process_2(const void *argument) {
    uint8_t          *bufRec;
    uint16_t         sizeRec;
    volatile mbox_t  *mbox;

// Waiting for the creation of the 0 to 1 one

    do {
        kern_switchFast();
    } while (kern_getMailboxDesc("Mailbox 0 to 1", &mbox) != KKERNNOERR);

    while (TRUE) {

// Receive a the message

        do {
            kern_switchFast();
        } while (kern_readMailbox(mailBox, &bufRec, &sizeRec) == KKERNNOPAC);
        iotx_printf(KURTO, "%s\n", bufRec);
        syos_free(bufRec);
    }
}
```

4.4. Statistics and μKernel performances

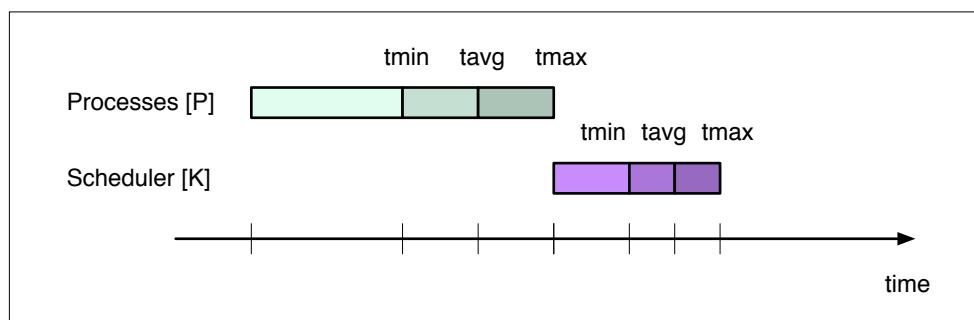
4.4.1. Statistics

The μKernel allows to measure the actual CPU time used by the processes during their **normal execution**, and by the μKernel during the **context switching**. This information, coupled with the number of times that the process was scheduled, allows to determine some important indications data about the real load of the CPU. Each process descriptor contains this information.

```
struct stts {
    uint64_t      oNbExecutions;           // # of time the process was scheduled
    uint64_t      oNbKernCalls;            // # of time that the kernel was called

    uint32_t      oAvstkSup;              // Available size in the sup. stack
    uint16_t      oTimePMin;              // Min CPU time used by the Process
    uint16_t      oTimePMax;              // Max CPU time used by the Process
    uint16_t      oTimePAvg;              // Avg CPU time used by the Process
    uint64_t      oTimePCum;              // Cum CPU time used by the Process
    uint16_t      oTimeKMin;              // Min CPU time used by the kernel
    uint16_t      oTimeKMax;              // Max CPU time used by the kernel
    uint16_t      oTimeKAvg;              // Avg CPU time used by the kernel
    uint64_t      oTimeECum;              // Cum CPU time used by the kernel
    uint16_t      oTimeEMin;              // Min CPU time used by the Exceptions
    uint16_t      oTimeEMax;              // Max CPU time used by the Exceptions
    uint16_t      oTimeEAvg;              // Avg CPU time used by the Exceptions
    uint64_t      oTimeEEcum;             // Cum CPU time used by the Exceptions
};

};
```



Total time CPU	$CPU_t = \sum_{i=0}^{i=nP} (P_{tavg_i} + K_{tavg_i}) \cdot nexec$	P_{tmin}	min. CPU time used by the process
Process time CPU	$PCPU[\%] = \frac{P_{tavg} \cdot nexec}{CPU_t} \cdot 100$	P_{tavg}	avg. CPU time used by the process
μKernel time CPU	$KCPU[\%] = \frac{K_{tavg} \cdot nexec}{CPU_t} \cdot 100$	P_{tmax}	max. CPU time used by the process
Process efficiency	$E[\%] = \frac{P_{tavg}}{P_{tavg} + K_{tavg}} \cdot 100$	K_{tmin}	min. CPU time used by the μKernel
		K_{tavg}	avg. CPU time used by the μKernel
		K_{tmax}	max. CPU time used by the μKernel
		$nexec$	nb. of time that the process was running

Sometimes it is useful to measure the CPU time used by a portion of code. The system call **kern_getTickCount** can be used for this purpose. Here is an example:

```
static void process_1(const void *argument) {
    uint64_t      time[2];
    uint32_t      duration;

    while (TRUE) {
        kern_getTickCount(&time[0]);           // ...
        routineToBeMeasured();                  // ...
        kern_getTickCount(&time[1]);           // ...
        duration = (uint32_t)(time[1] - time[0]); // Measure the time
        ...
    }
}
```

4.4.2. μKernel performances / measurements

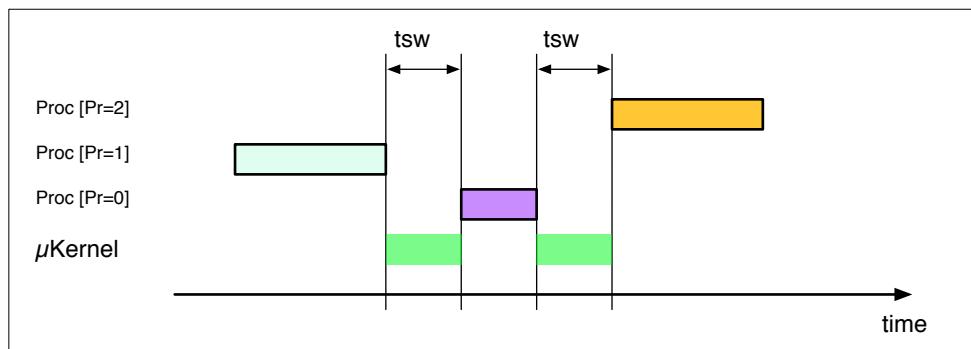
In order to obtain the best system performance, the time spent internally to the μKernel has to be as short as possible. Mainly, this time depends on:

1. Algorithms used for the different parts of the μKernel.
2. The frequency of the CPU.
3. The hardware implementation (bus size, memory cycle speed, DMA's, etc.).
4. The system interruption scheme and activity.

For the μKOS implementation the most important times used by the μKernel are: the **process switching time**, the Interruption Service Routine (**ISR**) **latency time**, the **event** and the **semaphore shuffling time**. The following performances have been measured on the CSEM Vision SoC IcyCAM running at 33-MHz and on a **STM32F407 & STM32F746** running at 168-MHz and at 216-MHz.

The process switching time

The μKernel is responsible for allocating the time slots to the different processes available in the execution list. When a condition imposing a context switching occurs (process timeout, synchronizations, etc.), then the μKernel determines which process has to obtain the CPU during the next slot of time (usually, it is the process having the highest priority, or, in case of many processes having the same priority, the one which was waiting for the CPU for a longer time).



Routine	Function	icyflex-1	Cortex-M4F (168-MHz)	Cortex-M7F (216-MHz)
		t[μs]	t[μs] (FPU) t[μs]	t[μs] (FPU) t[μs]
Save the context		2.5	0.16	0.3
sche_changeContext		10	2.9	2.9
	_nextAction			
	_getPriority			
	stat_statistic			
Restore the context		2.5	0.16	0.3
Total tsw		15	3.22	3.5
				1.42
				1.62

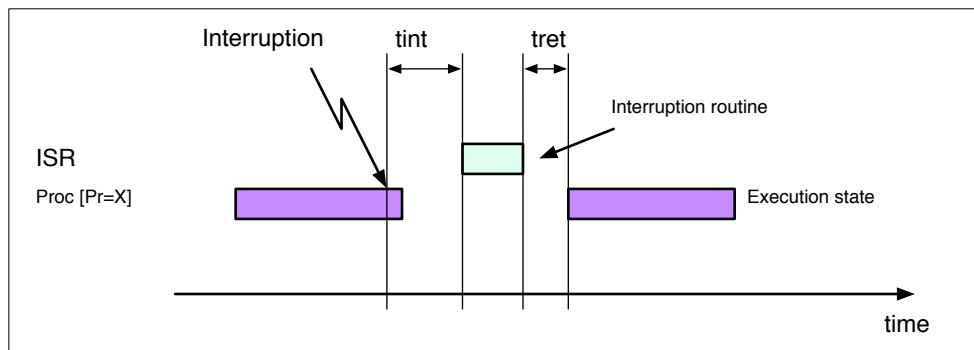
If necessary, the statistic functionality (handled by the routine `stat_statistic()`) can be disabled by a compilation flag. This allows to reduce the switching time in limited performances systems.

The ISR latency time

Interruption latency refers primarily to the software interrupt handling latency. This is the amount of time elapsed from the occurrence of an external interruption until its concrete servicing.

μKOS gives a highest priority to the peripheral interruptions in respect to the μKernel interruptions. This makes possible to obtain better reaction times and to deserve the needs of the peripherals faster. Of course, when necessary, the peripheral interruption routines can send messages via **synchro events** or **semaphores** to the μKernel. To ensure this reactivity, it is necessary to avoid the complete masking of all the interruption levels. In a normal operation, the interruption latency time is very short (just the necessary time to save the scratch registers, and execute the appropriate interruption code). Here is an example of interruption routine (from the `urt0` manager).

The first dashed area emphasizes the overhead code responsible for recovering the necessary information before executing the interruption function. The second dashed area corresponds to the code responsible for restoring the used registers and leave the interruption.

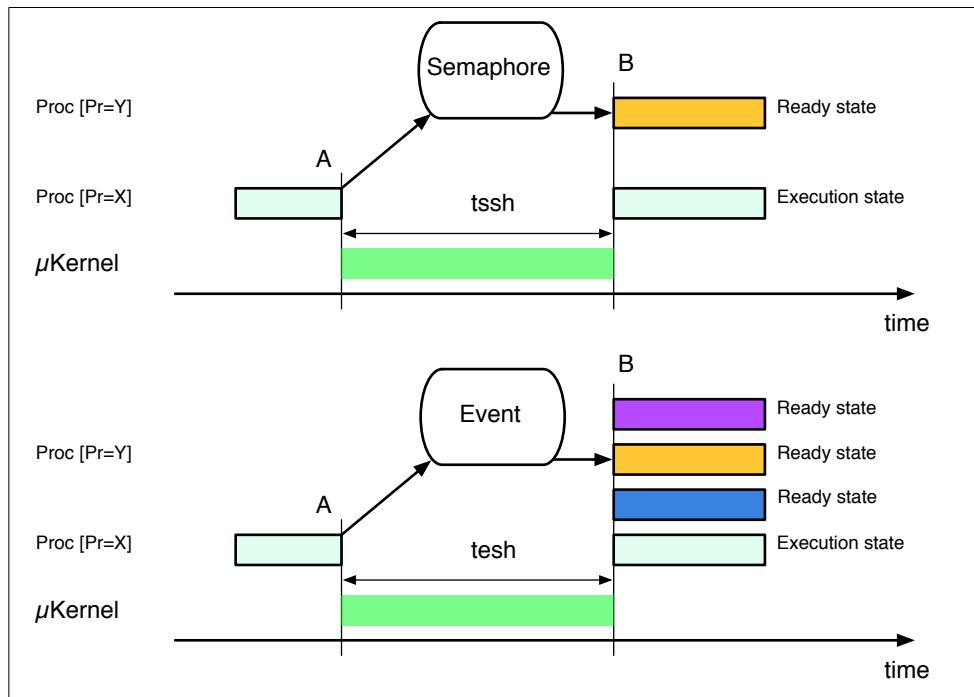


	icyflex-1 (33-MHz)		cortex-M4F (168-MHz)		cortex-M7F (216-MHz)	
	tint min [μs]	tret [μs]	tint min [μs]	tret [μs]	tint min [μs]	tret [μs]
Latency	0.5	0.5	>= 0.04	>= 0.04	>= 0.02	>= 0.02

It is evident that the latency time **tint** can be higher, if for any reason, the current process is under **INTERRUPTION_OFF** state.

The event and the semaphore shuffling time

In a real-time operating system **semaphores** are used to protect critical resources as well as for the synchronization mechanism. **Events** are used to permit fast synchronization between hardware (or process) and one or a group of processes. In both cases, the shuffling time is similar, and corresponds to the necessary amount of time for a process to communicate this synchronization and to place the on-hold processes inside the execution list.



	icyflex-1 (33-MHz)		cortex-M4F (168-MHz)		cortex-M7F (216-MHz)	
	tesh [μs]	tssh [μs]	tesh [μs]	tssh [μs]	tesh [μs]	tssh [μs]
Shuffling	22	22	6.5	6.5	4.9	4.9

Global μKernel overheads

Functions	icyflex-1 (33-MHz)		cortex-M4F (168-MHz)		cortex-M7F (216-MHz)	
	t [μs]	CPU	t [μs]	CPU [%]	t [μs]	CPU [%]
Interruption Fast (1-ms)	6	0.6	1.2	0.12	0.9	0.09
Timeout process (20-ms)	15	0.075	3.22	0.016	2.5	0.0123
Daemon Timeout processes (5-ms)	7	0.14	3.2	0.064	2.48	0.048
Daemon Stack sanity (500-ms) Stack size 800 long words	9	~ 0	3.5	~ 0	2.52	~ 0

```
// Example of interruption routine

_handle_urt0_interruptionUartRec:
    jsr    _saveScratchRegisters           ; Save the scratch registers
    nop
    sjsr  _stub_urt0_interruptionUartRec
    nop
    jsr    _restoreScratchRegisters       ; Restore the scratch registers
    nop
    rte
    nop

/*
 * Interruption RX
 * /////////////
 *
 */
void  stub_urt0_interruptionUartRec(void) {

    if (*UART_RXSTA & (1<<BRX_OVERRUN)) {
        *UART_RXSTA = (1<<BRX_OVERRUN);
        vStatus = KURT0EROVR;
    }
    ...

// Read the data and signal the event

    *vWRecBuffer++ = *UART_RX;
    ...

    if (vRXSema) kern_signalSemaphore(vSeHandleRX[0]);
}
```

4.5. Support of the multithread newlib

The use of the **newlib** in a multi-process environment requires to support the switching of the newlib's **_impure_ptr** associated to each process. The μKOS implementation defines two macros and two system calls to allow a correct newlib context switching.

```
#define      NLIB_PROCESS_INIT \
    static struct _reent      impure_data  __attribute__ ((aligned (8))); \
    _impure_ptr = &impure_data; \
    _REENT_INIT_PTR(_impure_ptr) \
    kern_setLocalPtr((void *)_impure_ptr); \


#define      NLIB_INIT \
    kern_setGlobalPtr((void *)&_impure_ptr);

void process_0(const void *argument) {

    NLIB_PROCESS_INIT;
    while (TRUE) {
        ...
    }
}

/*
 * main
 * ====
 *
 */
int main(void) {

    NLIB_INIT:           // Initialize the newlib _impure_ptr
    ...
    kern_createProcess(..., process_0, ...);
    return (0);
}
```

In the **main** program, the global **_impure_ptr** is passed to the μKernel by the macros **NLIB_INIT**. Each process defines and initializes its own data array by using the macros **NLIB_PROCESS_INIT**. During the context switching, the global pointer **_impure_ptr** previously passed in the μKernel is automatically set with the address of the array defined inside the process (**impure_data**).

4.6. Customization by stubs

The μKernel can be easily customized to other platforms or CPUs via a single **stub** responsible for handling all the low level parts: typically, timer interruptions, stack frame initializations and synchronizations via traps. μKOS μKernel exports 15 routines responsible for this customization.

```
void kernel_init(void)
• Some initializations.

void kernel_runKernel(void)
• Initialize all the timers (5-ms & 20-ms and time-stamp).
• Enable the time sharing and all the uKernel interruptions.

void kernel_loadOverlay(uint32_t idModule)
• Load an overlay process inside the overlay RAM.

void kernel_setLowPower(void)
• Set the low power mode.

void kernel_runPrecise(uint16_t time, void (*code)(void))
• Run a function precisely (1-μs resolution).

void kernel_initStackFrame(volatile proc_t *handle, const void *argument)
• Initialize and prepare the stack frame for a process.

void kernel_setUnixTime(uint32_t unixTime)
• Set the Unix time (1-s of resolution).

void kernel_getUnixTime(uint32_t *unixTime)
• Get the Unix time (1-s of resolution).

void kernel_timeStamp(uint64_t *timeStamp)
• Give a new time-stamp (1-us of resolution).

void kernel_getPC(const uint32_t *stackProcess, uint32_t *pc)
• Get the PC of a process.

void kernel_getFunctionName(const uint32_t pc, const char_t **function)
• Get the C function name.
```

```
void kernel_timeNext(void)
```

- Give a new time for a process.

```
void kernel_interruptionFast(void)
```

- Increment the tickcount variable (1-us of resolution).
- Verify the timeout condition of the suspended processes.

```
void kernel_switcher(void)
```

- Give another timeout to the process.
- Recover another context.

```
void kernel_message(void)
```

- Process the TRAP function.
- Recover another context.

4.6.1. The kernel model for the ARM Cortex M3

```
/*
; model_kernel_tim_1_2_3_4_ex0_ex2.
; =====

; -----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr                      $: Author of last commit
; $Rev::: 11                           $: Revision of last commit
; $Date::: 2016-10-22 21:00:43#$: Date of last commit
;
; Project:    uKOS
; Goal:       Model for controlling the "model_kernel_tim_1_2_3_4_ex0_ex2" device.
;
;           Tim 1 - 1-ms reference time
;           Tim 2 - 20-ms      process timeout
;           Tim 3 - 1-us free running timer
;           Tim 4 - user defined intervals for precise state-machines
;           ex0   - software interruption for message passing
;           ex2   - software interruption for preemption
;
;           This code has to support the following routines:
;           - kernel_init
;           - kernel_runKernel
;           - kernel_loadOverlay
;           - kernel_setLowPower
;           - kernel_runPrecise
;           - kernel_initStackFrame
```

```
;           - kernel_setUnixTime
;           - kernel_getUnixTime
;           - kernel_timeStamp
;           - kernel_timeNext
;           - kernel_interruptionFast
;           - kernel_switcher
;           - kernel_message
;           - kernel_getPC
;           - kernel_getFunctionName
;
; (c) 1992-2017, Franz Edo.
; -----
;
; Franz Edo.          _/ \_ / / \_ / \_ / \_ / \_
; 5-Route de Cheseaux  / / / / ,< / / / / \_ \_
; CH 1400 Cheseaux-Noréaz  / / / / | / / / / \_ / /
;                           \_,/_/ |_\_ / \_ / \_ / \_
; edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
; -----
*/
```

```
#define KFPRET1          1000000
#define KFINTT1          (1/KTTIMEFAST)
#define KPSCT1           ((KFREQUENCY_72/KFPRET1)-1)
#define KARRT1           ((KFPRET1/KFINTT1)-1)

#define KFPRET2          1000000
#define KFINTT2          (1/KTTIMESHARING)
#define KPSCT2           ((KFREQUENCY_72/KFPRET2)-1)
#define KARRT2           ((KFPRET2/KFINTT2)-1)

#define KFPRET3          1000000
#define KFINTT3          KTTIMESTATISTIC
#define KPSCT3           ((KFREQUENCY_72/KFPRET3)-1)
#define KARRT3           (-1)

extern void (*vExce_indIntVectors[KNBINTERRUPTIONS])(void);
extern void (*vKern_codeRoutine)(uint8_t state);
static volatile void (*vCodeRunPrecise)(void);
extern volatile proc_t *vKern_backwardProcess;
static volatile uint16_t vTimeStart = 0;
static volatile uint16_t vTimeStop = 0;
static volatile uint16_t vTimeLastStart = 0;
static volatile uint64_t vTiccountUs = 0;
static volatile uint64_t vTiccountMs = 0;
volatile uint32_t vKern_stackProcess;
volatile uint32_t vKern_message;

// Prototypes
// =====

static void _TIM4_IRQHandler(void);
static void _newContextTOU(void);
static void _newContextMSG(void);
static void _timeStart(void);
static void _timeStop(void);
static void _timeStamp(void);
```

```
/*
 * \brief kernel_init
 *
 * - Turn on the timers
 *
 */
void  kernel_init(void) {

    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;

    vUnixTime = 0;
}

/*
 * \brief kernel_runKernel
 *
 * - Initialize all the timers (1-ms & 20-ms and time-stamp)
 * - Enable the time sharing and all the uKernel interruptions
 *
 */
void  kernel_runKernel(void) {

    NVIC->IP[TIM1_UP_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM1_UP_IRQn & 0x1F));

    NVIC->IP[TIM2_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM2_IRQn & 0x1F));

    NVIC->IP[TIM4_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM4_IRQn & 0x1F));

    NVIC->IP[EXTI2_IRQn] = (KINT_LEVEL_ALL<<4);
    NVIC->ISER[0] |= (1<<(EXTI2_IRQn & 0x1F));

    NVIC->IP[EXTI0_IRQn] = (KINT_LEVEL_SWI<<4);
    NVIC->ISER[0] |= (1<<(EXTI0_IRQn & 0x1F));

// Timer 1 (1-ms)

    TIM1->PSC  = KPSCT1;
    TIM1->ARR  = KARRT1;
    TIM1->DIER = TIM_DIER_UIE;
    TIM1->CR1  |= TIM_CR1_CEN;
```

```
// Timer 2 (20-ms)

    TIM2->PSC  = KPSCT2;
    TIM2->ARR  = KARRT2;
    TIM2->DIER = TIM_DIER_UIE;
    TIM2->CR1 |= TIM_CR1_CEN;

// Timer 3 (free running timer for statistics)

    TIM3->PSC  = KPSCT3;
    TIM3->ARR  = KARRT3;
    TIM3->DIER = 0;
    TIM3->CR1 |= TIM_CR1_CEN;

// The software interruptions

    EXTI->IMR |= (1<<BKERNPREEMPTION);
    EXTI->IMR |= (1<<BKERNSWIMSG);
}

#endif (defined(__WITHOVLY__))
/*
 * \brief kernel_loadOverlay
 *
 * - Load an overlay process inside the overlay RAM
 *
 */
void  kernel_loadOverlay(uint32_t idModule) {

}
```

```
/*
 * \brief kernel_setLowPower
 *
 * - Set the low power mode
 *
 */
void  kernel_setLowPower(uint8_t mode) {

    switch (mode) {
        case KCPUPNORMAL:
        case KCPUPDEEP:
        case KCPUPHIBERNATE:
        case KCPUPSTOP:
        default: {

            // Waiting for an interruption

            WAITING_INTERRUPT;
            break;
        }
    }
}
```

```
/*
 * \brief kernel_runPrecise
 *
 * - Initialize the runPrecise function
 */
void  kernel_runPrecise(uint16_t time, void (*code)()) {

    vExce_indIntVectors[TIM4_IRQn] = aTIM4_IRQHandler;

    vCodeRunPrecise = code;
    switch (time) {
        case 0: {
            TIM4->DIER = 0;
            TIM4->CR1 &= ~TIM_CR1_CEN;
            break;
        }
        default: {
            TIM4->DIER = 0;
            TIM4->CR1 &= ~TIM_CR1_CEN;
            TIM4->SR  &= ~TIM_SR_UIF;

            TIM4->PSC  = (uint16_t)((KFREQUENCY_APB2/1000000)-1);
            TIM4->ARR  = time - 1;
            TIM4->DIER = TIM_DIER_UIE;
            TIM4->CR1 |= TIM_CR1_CEN;
            break;
        }
    }
}

static void  aTIM4_IRQHandler(void) {
    void  (*code)(void);

    // INT acknowledge

    if (TIM4->SR & TIM_SR_UIF)
        TIM4->SR &= ~TIM_SR_UIF;

    code = (void (*)(void))vCodeRunPrecise;
    code();
}
```

```
/*
 * \brief kernel_initStackFrame
 *
 * - Initialize and prepare the stack frame for a process
 *
 */
void  kernel_initStackFrame(volatile proc_t *handle, const void *argument) {
    volatile      uint32_t      *stackFrame, i;

    // Fill the stack with a "magic" pattern (for statistic)

    stackFrame = handle->oSpecification.oStkSup;                                //
    for (i = 0; i < handle->oSpecification.oSzStk; i++)                         //
        *--stackFrame = (('u'<<24) | ('K'<<16) | ('O'<<8) | 'S');           //

    // Prepare the stack frame

    stackFrame = handle->oSpecification.oStkSup;                                //
    *--stackFrame = 0x01000000;                                                 // xPSR
    *--stackFrame = (uint32_t)handle->oSpecification.oCode;                      // PC
    *--stackFrame = 0xFFFFFFFFF9;                                                 // LR
    *--stackFrame = 0x12121212;                                                 // r12
    *--stackFrame = 0x03030303;                                                 // r3
    *--stackFrame = 0x02020202;                                                 // r2
    *--stackFrame = 0x01010101;                                                 // r1
    *--stackFrame = (uint32_t)argument;                                           // r0
    *--stackFrame = 0x11111111;                                                 // r11
    *--stackFrame = 0x10101010;                                                 // r10
    *--stackFrame = 0x09090909;                                                 // r9
    *--stackFrame = 0x08080808;                                                 // r8
    *--stackFrame = 0x07070707;                                                 // r7
    *--stackFrame = 0x06060606;                                                 // r6
    *--stackFrame = 0x05050505;                                                 // r5
    *--stackFrame = 0x04040404;                                                 // r4
    *--stackFrame = (KINT_IMASK_ALL<<4);                                     // basepri
    *--stackFrame = 0xFFFFFFFFF9;                                               // exc ret

    handle->ooSpecification.oStkSup = (uint32_t *)stackFrame;
}

/*
 * \brief kernel_setUnixTime
 *
 * - Set the UNIX absolute time
 *
 */
void  kernel_setUnixTime(uint32_t unixTime) {

    vUnixTime = unixTime;
}
```

```
/*
 * \brief kernel_getUnixTime
 *
 * - Get the UNIX absolute time
 *
 */
void    kernel_getUnixTime(uint32_t *unixTime) {

    *unixTime = vUnixTime;
}

/*
 * \brief kernel_timeStamp
 *
 * - Give a new 64-bit time-stamp (1-us of resolution)
 *   - 0x0000000000000000 to 0xFFFFFFFFFFFFFF incremental counter
 *
 */
void    kernel_timeStampUs(uint64_t *timeStamp) {

    _timeStamp();
    *timeStamp = vTiccount;
}

/*
 * \brief kernel_timeNext
 *
 * - Give a new time for a process
 *
 */
void    kernel_timeNext(void) {

    TIM2->CNT    = 0;
    TIM2->ARR    = KARRT2;
    TIM2->SR     &= ~TIM_SR UIF;
    TIM2->CR1    |=  TIM_CR1_CEN;
}
```

```
/*
 * \brief kernel_getPC
 *
 * - Get a value of the PC
 *
 */
void  kernel_getPC(const uint32_t *stackProcess, uint32_t *pc) {
    uint8_t      pcOffset = 1;

// uKOS stackframe:
//
// stmdb          r0!,{r14}           -> pcOffset = 1
// stmdb          r0!,{r1,r4-r11}     -> pcOffset += 1 + ((11 - 4) + 1)
//
// Automatic stacked
//
//          r3-r0           -> pcOffset += ((3 - 0) + 1)
//          r12             -> pcOffset += 1
//          lr (r14)         -> pcOffset += 1
//          PC

//
//          r1,      r11 to r4,       r3 to r0,      r12,   r14
//          --  -----  -----  ---  ---
//          pcOffset += 1 + ((11 - 4) + 1) + ((3 - 0) + 1) + 1 + 1;
*pc = (stackProcess[pcOffset]);
}
```

```
/*
 * \brief kernel_getFunctionName
 *
 * - Get the function name prepended by the gcc compiler
 *   (extraction using the option "-mpoke-function-name"
 *
 * t0
 *     .ascii "arm_poke_function_name", 0
 *     .align
 *
 * t1
 *     .word 0xffff00000 + (t1 - t0)
 *     arm_poke_function_name
 *     mov    ip, sp
 *     stmdfd sp!, {fp, ip, lr, pc}
 *     sub    fp, ip, #4
 *
 */
void  kernel_getFunctionName(const uint32_t pc, const char_t **function) {
    uint16_t      off;
    uint32_t      nameLen;
    const uint32_t      *ptr;

    ptr = (const uint32_t *)((pc + 2) & (~0x3U));

    for (off = 1; (off < 16*1024) && ((uint32_t)&ptr[-off] > 0x0); ++off) {
        if ((ptr[-off] & 0xFFFFFFF00) == 0xFF000000) {
            nameLen = ptr[-off] & 0xFFU;
            *function = &((const char_t*)&ptr[-off])[-nameLen];
            return;
        }
    }
    *function = NULL;
}
```

```
// Local routines
// =====

/*
 * \brief TIM1_UP_IRQHandler
 *
 * - Fine time interruption
 * - Increment the tic-count 1-us variable
 * - Verify the timeout condition of the suspended processes
 *
 */
void    TIM1_UP_IRQHandler(void) {
    static uint32_t      counter = 0;

// INT acknowledge
// Tic_count++ and verify the timeout conditions
// Increment the UNIX time seconds

    if ((TIM1->SR & TIM_SR UIF) != 0) {
        TIM1->SR &= ~TIM_SR UIF;
    }

    scheCallBackFast();
    if (counter++ % (1000/KTIMEUNIT) == 0) {
        vUnixTime++;
    }
    _timeStamp();
}
```

```

/*
 * \brief EXTI2_IRQHandler
 *
 * - Preemption
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
*      IOFF
*
*      ->    xPSR
*      ->    PC
*      ->    LR
*      ->    R12
*      ->    R3..R0           Block stacked by an exception
*
*      ->    R11..R4
*      ->    BASEPRI
*      ->    LR(R14)          Block stacked manually
*
* !!! Do not generate prologue/epilogue sequences
*
*/
// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif

void  EXTI2_IRQHandler(void)__attribute__((naked)) __attribute__((optimize("Os")));
void  EXTI2_IRQHandler(void) {

// Interruption ACK

    EXTI->PR |= (1<<BKERNPREEMPTION);

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid      i              \n"    //
        "mrs       r0,msp          \n"    // r0 = stack
        "mrs       r1,basepri      \n"    // r1 = basepri
    )
}

```

```
"stmdb      r0!,{r1,r4-r11}    \n"    // Save the register list
"stmdb      r0!,{r14}          \n"    // Save the EXCEPTION return
"msr       msp,r0            \n"    //
"str       r0,%0             \n"    // Save the stack of the old process
"cpsie     i                 \n"    //
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

// Change the context due to a timeout

_newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
"cpsid     i                 \n"    //
"ldr       r0,%0             \n"    // Recover the stack of the new process
"ldmia    r0!,{r14}           \n"    // Restore the EXCEPTION return
"ldmia    r0!,{r1,r4-r11}     \n"    // Restore the register list
"msr       msp,r0            \n"    // New stack
"msr       basepri,r1        \n"    // Restore the basepri
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

// Return

__asm__ volatile (
"cpsie     i                 \n"    //
"dmb      \n"    //
"dsb      \n"    //
"isb      \n"    //
"bx       lr                \n"    // Return
);
}
```

```

/*
 * \brief TIM2_IRQHandler
 *
 * - Time slice timeout
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
*      IOFF
*
*      ->    xPSR
*      ->    PC
*      ->    LR
*      ->    R12
*      ->    R3..R0      Block stacked by an exception
*
*      ->    R11..R4
*      ->    BASEPRI
*      ->    LR(R14)     Block stacked manually
*
* !!! Do not generate prologue/epilogue sequences
*
*/
// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif
void  TIM2_IRQHandler(void) __attribute__ ((naked)) __attribute__ ((optimize("Os")));
void  TIM2_IRQHandler(void) {

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid    i          \n"      // 
        "mrs     r0,msp      \n"      // r0 = stack
        "mrs     r1,basepri \n"      // r1 = basepri
        "stmdb   r0!,{r1,r4-r11} \n"  // Save the register list
        "stmdb   r0!,{r14}    \n"      // Save the EXCEPTION return
        "msr     msp,r0      \n"      //
        "str     r0,%0      \n"      // Save the stack of the old process
        "cpsie   i          \n"      // 
    );
}

```

```
:  
: "m" (vKern_stackProcess)  
: KSAVEREGISTERS  
);  
  
// Change the context due to a timeout  
  
_newContextTOU();  
  
// Restore the registers r4..r11 and the basepri  
  
__asm__ volatile (  
"cpsid      i          \n"    //  
"ldr        r0,%0      \n"    // Recover the stack of the new process  
"ldmia     r0!,{r14}    \n"    // Restore the EXCEPTION return  
"ldmia     r0!,{r1,r4-r11}\n"   // Restore the register list  
"msr       msp,r0      \n"    // New stack  
"msr       basepri,r1   \n"    // Restore the basepri  
:  
: "m" (vKern_stackProcess)  
: KSAVEREGISTERS  
);  
  
// Return  
  
__asm__ volatile (  
"cpsie      i          \n"    //  
"dmb        \n"      //  
"dsb        \n"      //  
"isb        \n"      //  
"bx         lr        \n"    // Return  
);  
}
```

```
/*
 * \brief _newContextTOU
 *
 * - Change the context due to a timeout
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextTOU(void) {

    _timeStop();
    if ((TIM2->SR & TIM_SR UIF) != 0) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackSlow();
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess,
                   vTimeStart,
                   vTimeStop,
                   vTimeLastStart,
                   vTimeException);
    vTimeException = 0;
#endif
}
```

```
/*
 * \brief EXTI0_IRQHandler
 *
 * - TRAP
 * - Save the context
 * - Save the context stack
 * - Process the message sent
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          IOFF
 *
 *      -> Message      +36
 *      -> Message      +32
 *
 *      -> xPSR
 *      -> PC
 *      -> LR
 *      -> R12
 *      -> R3..R0      Block stacked by an exception
 *
 *      -> R11..R4
 *      -> BASEPRI
 *      -> LR(R14)      Block stacked manually
 *
 * !!! Do not generate prologue/epilogue sequences
 *
 */
void  EXTI0_IRQHandler(void)__attribute__((naked)) __attribute__((optimize("Os")));
void  EXTI0_IRQHandler(void) {

    // Interruption ACK

    EXTI->PR |= (1<<BKERNNSWIMSG);

    // Recover the swi message: r0 contains the stack

    __asm__ volatile (
        "cpsid    i                  \n"    //
        "mrs     r0,msp              \n"    // r0 = stack
        "add     r1,r0,#36            \n"    // r1 = message address
        "ldr     r1,[r1]              \n"    // r1 = message
        "str     r1,%0                \n"    // Save it on the vKern_message
        :
        : "m"  (vKern_message)
        : KSAVEREGISTERS
    );
}
```

```
// Save the registers r4..r11 and the basepri
// r0 contains the stack

__asm__ volatile (
    "mrs          r1,basepri           \n"    // r1 = basepri
    "stmdb        r0!,{r1,r4-r11}     \n"    // Save the register list
    "stmdb        r0!,{r14}           \n"    // Save the EXCEPTION return
    "msr          msp,r0             \n"    //
    "str          r0,%0              \n"    // Save the stack of the old process
    "cpsie        i                 \n"    //
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
);

// Change the context due to a message

_newContextMSG();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
    "cpsid        i                 \n"    //
    "ldr          r0,%0              \n"    // Recover the stack of the new process
    "ldmia       r0!,{r14}           \n"    // Restore the EXCEPTION return
    "ldmia       r0!,{r1,r4-r11}     \n"    // Restore the register list
    "msr          msp,r0             \n"    // New stack
    "msr          basepri,r1         \n"    // Restore the basepri
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
);

// Return

__asm__ volatile (
    "cpsie        i                 \n"    //
    "dmb          \n"                //
    "dsb          \n"                //
    "isb          \n"                //
    "bx          lr                 \n"    // Return
);
}
```

```
/*
 * \brief _newContextMSG
 *
 * - Change the context due to a message
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextMSG(void) {

    _timeStop();
    if ((TIM2->SR & TIM_SR UIF) != 0) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackTrap(vKern_message);
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess,
                   vTimeStart,
                   vTimeStop,
                   vTimeLastStart,
                   vTimeException);
    vTimeException = 0;
#endif
}
```

```
/*
 * \brief _timeStart
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStart(void) {

    vTimeLastStart = vTimeStart;
    vTimeStart      = TIM3->CNT;
}

/*
 * \brief _timeStop
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStop(void) {

    vTimeStop = TIM3->CNT;
}

/*
 * \brief _timeStamp
 *
 * - Maintain a 64-bit timer with 1-us of resolution
 *
 */
static void _timeStamp(void) {
    uint16_t      newTime;
    static uint16_t oldTime = 0;

    INTERRUPTION_OFF;
    newTime = TIM3->CNT;
    if (newTime < oldTime) vTiccount += 0x10000;

    vTiccount = (vTiccount & 0xFFFFFFFFFFFF0000) | newTime;
    oldTime = newTime;
    INTERRUPTION_RESTORED;
}
```

4.6.2. The kernel model for the ARM Cortex M4

```
/*
; model_kernel_tim_1_2_3_4_ex2_ex3.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author:: efr          $: Author of last commit
; $Rev::: 11           $: Revision of last commit
; $Date::: 2016-10-22 21:00:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        Model for controlling the "model_kernel_tim_1_2_3_4_ex2_ex3" device.
;
;             Tim 1 - 1-ms reference time
;             Tim 2 - 20-ms      process timeout
;             Tim 3 - 1-us free running timer
;             Tim 4 - user defined intervals for precise state-machines
;             ex2   - software interruption for preemption
;             ex3   - software interruption for message passing
;
;             This code has to support the following routines:
;             - kernel_init
;             - kernel_runKernel
;             - kernel_loadOverlay
;             - kernel_setLowPower
;             - kernel_runPrecise
;             - kernel_initStackFrame
;             - kernel_setUnixTime
;             - kernel_getUnixTime
;             - kernel_timeStamp
;             - kernel_timeNext
;             - kernel_interruptionFast
;             - kernel_switcher
;             - kernel_message
;             - kernel_getPC
;             - kernel_getFunctionName
;
```

```
; (c) 1992-2017, Franzi Edo.  
;  
;  
; Franzi Edo.          _ _ / / / _ \ _ /  
; 5-Route de Cheseaux  / / / , < / / / / \ _ \  
; CH 1400 Cheseaux-Noréaz / / / / | / / / / _ / /  
;                           \_,/_/ |_\_ / / _ / /  
; edo.franzi@ukos.ch  
  
;  
; This program is free software: you can redistribute it and/or modify  
; it under the terms of the GNU Affero General Public License as published by  
; the Free Software Foundation, either version 3 of the License, or  
; (at your option) any later version.  
  
;  
; This program is distributed in the hope that it will be useful,  
; but WITHOUT ANY WARRANTY; without even the implied warranty of  
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
; GNU Affero General Public License for more details.  
  
;  
; You should have received a copy of the GNU Affero General Public License  
; along with this program. If not, see <http://www.gnu.org/licenses/>.  
;  
;-----  
*/
```

```
#define KFPRET1 1000000
#define KFINTT1 (1/KTTIMEFAST)
#define KPSCT1 ((KFREQUENCY_84/KFPRET1)-1)
#define KARRT1 ((KFPRET1/KFINTT1)-1)

#define KFPRET2 1000000
#define KFINTT2 (1/KTTIMESHARING)
#define KPSCT2 ((KFREQUENCY_84/KFPRET2)-1)
#define KARRT2 ((KFPRET2/KFINTT2)-1)

#define KFPRET3 1000000
#define KFINTT3 KTTIMESTATISTIC
#define KPSCT3 ((KFREQUENCY_84/KFPRET3)-1)
#define KARRT3 (-1)

extern void (*vExce_indIntVectors[KNBINTERRUPTIONS])(void);
extern void (*vKern_codeRoutine)(uint8_t state);
static volatile void (*vCodeRunPrecise)(void);
extern volatile proc_t *vKern_backwardProcess;
static volatile uint16_t vTimeStart = 0;
static volatile uint16_t vTimeStop = 0;
static volatile uint16_t vTimeLastStart = 0;
static uint64_t vTickcount = 0;
static uint32_t vUnixTime;
volatile uint32_t vKern_stackProcess;
volatile uint32_t vKern_message;

// Prototypes
// =====

static void _TIM4_IRQHandler(void);
static void _newContextTOU(void);
static void _newContextMSG(void);
static void _timeStart(void);
static void _timeStop(void);
static void _timeStamp(void);
```

```
/*
 * \brief kernel_init
 *
 * - Turn on the timers
 *
 */
void  kernel_init(void) {

    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;

    vUnixTime = 0;
}

/*
 * \brief kernel_runKernel
 *
 * - Initialize all the timers (1-ms & 20-ms and time-stamp)
 * - Enable the time sharing and all the uKernel interruptions
 *
 */
void  kernel_runKernel(void) {

    NVIC->IP[TIM1_UP_TIM10_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM1_UP_TIM10_IRQn & 0x1F));

    NVIC->IP[TIM2_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM2_IRQn & 0x1F));

    NVIC->IP[TIM4_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM4_IRQn & 0x1F));

    NVIC->IP[EXTI2_IRQn] = (KINT_LEVEL_ALL<<4);
    NVIC->ISER[0] |= (1<<(EXTI2_IRQn & 0x1F));

    NVIC->IP[EXTI3_IRQn] = (KINT_LEVEL_SWI<<4);
    NVIC->ISER[0] |= (1<<(EXTI3_IRQn & 0x1F));

// Timer 1 (1-ms)

    TIM1->PSC  = KPSCT1;
    TIM1->ARR  = KARRT1;
    TIM1->DIER = TIM_DIER_UIE;
    TIM1->CR1  |= TIM_CR1_CEN;
```

```
// Timer 2 (20-ms)

    TIM2->PSC  = KPSCT2;
    TIM2->ARR  = KARRT2;
    TIM2->DIER = TIM_DIER_UIE;
    TIM2->CR1 |= TIM_CR1_CEN;

// Timer 3 (free running timer for statistics)

    TIM3->PSC  = KPSCT3;
    TIM3->ARR  = KARRT3;
    TIM3->DIER = 0;
    TIM3->CR1 |= TIM_CR1_CEN;

// The software interruptions

    EXTI->IMR |= (1<<BKERNPREEMPTION);
    EXTI->IMR |= (1<<BKERNSWIMSG);
}

#endif (defined(__WITHOVLY__))
/*
 * \brief kernel_loadOverlay
 *
 * - Load an overlay process inside the overlay RAM
 *
 */
void  kernel_loadOverlay(uint32_t idModule) {

}
```

```
/*
 * \brief kernel_setLowPower
 *
 * - Set the low power mode
 *
 */
void  kernel_setLowPower(uint8_t mode) {

    switch (mode) {
        case KCPUPNORMAL:
        case KCPUPDEEP:
        case KCPUPHIBERNATE:
        case KCPUPSTOP:
        default: {

            // Waiting for an interruption

            WAITING_INTERRUPT;
            break;
        }
    }
}
```

```
/*
 * \brief kernel_runPrecise
 *
 * - Initialize the runPrecise function
 */
void  kernel_runPrecise(uint16_t time, void (*code)()) {

    vExce_indIntVectors[TIM4_IRQn] = aTIM4_IRQHandler;

    vCodeRunPrecise = code;
    switch (time) {
        case 0: {
            TIM4->DIER = 0;
            TIM4->CR1 &= ~TIM_CR1_CEN;
            break;
        }
        default: {
            TIM4->DIER = 0;
            TIM4->CR1 &= ~TIM_CR1_CEN;
            TIM4->SR  &= ~TIM_SR_UIF;

            TIM4->PSC  = (uint16_t)((KFREQUENCY_APB2/1000000)-1);
            TIM4->ARR  = time - 1;
            TIM4->DIER = TIM_DIER_UIE;
            TIM4->CR1 |= TIM_CR1_CEN;
            break;
        }
    }
}

static void  aTIM4_IRQHandler(void) {
    void  (*code)(void);

    // INT acknowledge

    if (TIM4->SR & TIM_SR_UIF)
        TIM4->SR &= ~TIM_SR_UIF;

    code = (void (*)(void))vCodeRunPrecise;
    code();
}
```

```
/*
 * \brief kernel_initStackFrame
 *
 * - Initialize and prepare the stack frame for a process
 */
void  kernel_initStackFrame(volatile proc_t *handle, const void *argument) {
    volatile      uint32_t      *stackFrame, i;

    // Fill the stack with a "magic" pattern (for statistic)

    stackFrame = handle->oSpecification.oStkSup;                                //
    for (i = 0; i < handle->oSpecification.oSzStk; i++)                         //
        *--stackFrame = (('u'<<24) | ('K'<<16) | ('O'<<8) | 'S');           //

    // Prepare the stack frame

    stackFrame = handle->oSpecification.oStkSup;                                //
    *--stackFrame = 0x01000000;                                                 // xPSR
    *--stackFrame = (uint32_t)handle->oSpecification.oCode;                      // PC
    *--stackFrame = 0xFFFFFFFFFD;                                                // LR
    *--stackFrame = 0x12121212;                                                 // r12
    *--stackFrame = 0x03030303;                                                 // r3
    *--stackFrame = 0x02020202;                                                 // r2
    *--stackFrame = 0x01010101;                                                 // r1
    *--stackFrame = (uint32_t)argument;                                           // r0
    *--stackFrame = 0x11111111;                                                 // r11
    *--stackFrame = 0x10101010;                                                 // r10
    *--stackFrame = 0x09090909;                                                 // r9
    *--stackFrame = 0x08080808;                                                 // r8
    *--stackFrame = 0x07070707;                                                 // r7
    *--stackFrame = 0x06060606;                                                 // r6
    *--stackFrame = 0x05050505;                                                 // r5
    *--stackFrame = 0x04040404;                                                 // r4
    *--stackFrame = (KINT_IMASK_ALL<<4);                                     // basepri
    *--stackFrame = 0xFFFFFFFFFD;                                              // exc ret

    handle->oSpecification.oStkSup = (uint32_t *)stackFrame;
}
```

```
/*
 * \brief kernel_setUnixTime
 *
 * - Set the UNIX absolute time
 *
 */
void  kernel_setUnixTime(uint32_t unixTime) {

    vUnixTime = unixTime;
}

/*
 * \brief kernel_getUnixTime
 *
 * - Get the UNIX absolute time
 *
 */
void  kernel_getUnixTime(uint32_t *unixTime) {

    *unixTime = vUnixTime;
}

/*
 * \brief kernel_timeStamp
 *
 * - Give a new 64-bit time-stamp (1-us of resolution)
 *   - 0x0000000000000000 to 0xFFFFFFFFFFFFFF incremental counter
 *
 */
void  kernel_timeStamp(uint64_t *timeStamp) {

    _timeStamp();
    *timeStamp = vTiccount;
}

/*
 * \brief kernel_timeNext
 *
 * - Give a new time for a process
 *
 */
void  kernel_timeNext(void) {

    TIM2->CNT  = 0;
    TIM2->ARR  = KARRT2;
    TIM2->SR   &= ~TIM_SR UIF;
    TIM2->CR1 |= TIM_CR1_CEN;
}
```

```
/*
 * \brief kernel_getPC
 *
 * - Get a value of the PC
 *
 */
void  kernel_getPC(const uint32_t *stackProcess, uint32_t *pc) {
    uint8_t          pcOffset = 1;
    uint32_t         lr;

// uKOS stackframe:
// 
// stmdb          r0!,{r14}           -> pcOffset = 1
// 
// if the fpu was used
// vstmdbeq       r0!,{s16-s31}      -> pcOffset += ((31 - 16) + 1)
// 
// stmdb          r0!,{r1,r4-r11}     -> pcOffset += 1 + ((11 - 4) + 1)
// 
// Automatic stacked
// 
//          r3-r0           -> pcOffset += ((3 - 0) + 1)
//          r12             -> pcOffset += 1
//          lr (r14)        -> pcOffset += 1
//          PC

    lr = stackProcess[0];

// If the fpu was used                         s31 to s16
// 
// if ((lr & (1<<4)) == 0) { pcOffset += ((31 - 16) + 1); }

//          r1,          r11 to r4,          r3 to r0,          r12,   r14
//          --          -----          -----          ---   ---
// pcOffset += 1      + ((11 - 4) + 1)      + ((3 - 0) + 1)      + 1      + 1;
*pc = (stackProcess[pcOffset]);
}
```

```
/*
 * \brief kernel_getFunctionName
 *
 * - Get the function name prepended by the gcc compiler
 *   (extraction using the option "-mpoke-function-name"
 *
 * t0
 *     .ascii "arm_poke_function_name", 0
 *     .align
 *
 * t1
 *     .word 0xffff00000 + (t1 - t0)
 *     arm_poke_function_name
 *     mov    ip, sp
 *     stmdfd sp!, {fp, ip, lr, pc}
 *     sub    fp, ip, #4
 *
 */
void  kernel_getFunctionName(const uint32_t pc, const char_t **function) {
    uint16_t      off;
    uint32_t      nameLen;
    const uint32_t      *ptr;

    ptr = (const uint32_t *)((pc + 2) & (~0x3U));

    for (off = 1; (off < 16*1024) && ((uint32_t)&ptr[-off] > 0x0); ++off) {
        if ((ptr[-off] & 0xFFFFFFF00) == 0xFF000000) {
            nameLen = ptr[-off] & 0xFFU;
            *function = &((const char_t*)&ptr[-off])[-nameLen];
            return;
        }
    }
    *function = NULL;
}
```

```
// Local routines
// =====

/*
 * \brief TIM1_UP_TIM10_IRQHandler
 *
 * - Fine time interruption
 * - Increment the tic-count 1-us variable
 * - Verify the timeout condition of the suspended processes
 *
 */
void    TIM1_UP_TIM10_IRQHandler(void) {
    static uint32_t      counter = 0;

// INT acknowledge
// Tic_count++ and verify the timeout conditions
// Increment the UNIX time seconds

    if ((TIM1->SR & TIM_SR UIF) != 0) {
        TIM1->SR &= ~TIM_SR UIF;
    }

    scheCallBackFast();
    if (counter++ % (1000/KTIMEUNIT) == 0) {
        vUnixTime++;
    }
    _timeStamp();
}
```

```
/*
 * \brief EXTI2_IRQHandler
 *
 * - Preemption
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          Normal stack frame      Stack frame with fpu
 *          IOFF                  IOFF
 *
 *          ->                   FPSCR
 *          ->                   S15..S0
 *          ->      xPSR           xPSR
 *          ->      PC              PC
 *          ->      LR(R14)        LR(R14)
 *          ->      R12             R12
 *          ->      R3..R0         R3..R0      Block stacked by an exception
 *
 *          ->      R11..R4        R11..R4
 *          ->      BASEPRI        BASEPRI
 *          ->                   S31..S16
 *          ->      LR(R14)        LR(R14)      Block stacked manually
 *
 *
 * !!! Do not generate prologue/epilogue sequences
 */
// Test for GCC version for adapting the assembler context code

#if      (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif

void  EXTI2_IRQHandler(void)__attribute__((naked)) __attribute__((optimize("Os")));
void  EXTI2_IRQHandler(void) {

// Interruption ACK

    EXTI->PR |= (1<<BKERNPREEMPTION);

// Save the registers r4..r11 and the basepri
// r0 contains the stack
```

```

__asm__ volatile (
    "cpsid      i          \n"      //
    "mrs       r0,psp      \n"      // r0 = stack
    "mrs       r1,basepri  \n"      // r1 = basepri
    "stmdb     r0!,{r1,r4-r11} \n"   // Save the register list
    "tst       r14,#0x10   \n"      //
    "it        eq         \n"      //
    "vstmdbeq  r0!,{s16-s31} \n"   // If used, save the fp registers
    "stmdb     r0!,{r14}    \n"      // Save the EXCEPTION return
    "msr       psp,r0     \n"      //
    "str       r0,%0      \n"      // Save the stack of the old process
    "cpsie    i          \n"      //
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Change the context due to a timeout

_newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
    "cpsid      i          \n"      //
    "ldr       r0,%0      \n"      // Recover the stack of the new process
    "ldmia    r0!,{r14}    \n"      // Recover the EXCEPTION return
    "tst       r14,#0x10   \n"      //
    "it        eq         \n"      //
    "vldmiaeq  r0!,{s16-s31} \n"   // If used, restore the fp registers
    "ldmia    r0!,{r1,r4-r11} \n"   // Restore the register list
    "msr       psp,r0     \n"      // New stack
    "msr       basepri,r1  \n"      // Restore the basepri
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Return

__asm__ volatile (
    "cpsie    i          \n"      //
    "dmb      \n"      //
    "dsb      \n"      //
    "isb      \n"      //
    "bx       lr        \n"      // Return
);
}

```

```

/*
 * \brief TIM2_IRQHandler
 *
 * - Time slice timeout
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *

*          Normal stack frame      Stack frame with fpu
*          IOFF                  IOFF
*
*          ->                   FPSCR
*          ->                   S15..S0
*          ->      xPSR           xPSR
*          ->      PC              PC
*          ->      LR(R14)        LR(R14)
*          ->      R12             R12
*          ->      R3..R0         R3..R0      Block stacked by an exception
*
*          ->      R11..R4        R11..R4
*          ->      BASEPRI        BASEPRI
*          ->                   S31..S16
*          ->      LR(R14)        LR(R14)      Block stacked manually
*
* !!! Do not generate prologue/epilogue sequences
*/
// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define    KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define    KSAVEREGISTERS
#endif
void  TIM2_IRQHandler(void) __attribute__ ((naked)) __attribute__ ((optimize("Os")));
void  TIM2_IRQHandler(void) {

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid    i                  \n"    //
        "mrs     r0,psp            \n"    // r0 = stack
        "mrs     r1,basepri        \n"    // r1 = basepri
        "stmdb   r0!,{r1,r4-r11}    \n"    // Save the register list
}

```

```

    "tst      r14,#0x10          \n"    //
    "it       eq                 \n"    //
    "vstmdbeq r0!,{s16-s31}     \n"    // If used, save the fp registers
    "stmdb   r0!,{r14}          \n"    // Save the EXCEPTION return
    "msr     psp,r0            \n"    //
    "str     r0,%0              \n"    // Save the stack of the old process
    "cpsie   i                 \n"    //
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Change the context due to a timeout

    _newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
    "cpsid   i                  \n"    //
    "ldr     r0,%0              \n"    // Recover the stack of the new process
    "ldmia   r0!,{r14}          \n"    // Recover the EXCEPTION return
    "tst     r14,#0x10          \n"    //
    "it      eq                 \n"    //
    "vldmiaeq r0!,{s16-s31}     \n"    // If used, restore the fp registers
    "ldmia   r0!,{r1,r4-r11}    \n"    // Restore the register list
    "msr     psp,r0            \n"    // New stack
    "msr     basepri,r1        \n"    // Restore the basepri
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Return

__asm__ volatile (
    "cpsie   i                  \n"    //
    "dmb    "
    "dsb    "
    "isb    "
    "bx     lr                 \n"    // Return
    );
}

```

```
/*
 * \brief _newContextTOU
 *
 * - Change the context due to a timeout
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextTOU(void) {

    _timeStop();
    if ((TIM2->SR & TIM_SR UIF) !=0) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackSlow();
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess,
                   vTimeStart,
                   vTimeStop,
                   vTimeLastStart,
                   vTimeException);
    vTimeException = 0;
#endif
}
```

```
/*
 * \brief EXTI3_IRQHandler
 *
 * - TRAP
 * - Save the context
 * - Save the context stack
 * - Process the message sent
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          Normal stack frame      Stack frame with fpu
 *          IOFF                  IOFF
 *
 *          ->      Message      +36    +108
 *          ->      Message      +32    +104
 *
 *          ->                      FPSCR
 *          ->                      S15..S0
 *          ->      xPSR        xPSR
 *          ->      PC           PC
 *          ->      LR(R14)     LR(R14)
 *          ->      R12          R12
 *          ->      R3..R0      R3..R0      Block stacked by an exception
 *
 *          ->      R11..R4      R11..R4
 *          ->      BASEPRI     BASEPRI
 *          ->                      S31..S16
 *          ->      LR(R14)     LR(R14)      Block stacked manually
 *
 * !!! Do not generate prologue/epilogue sequences
 */
void  EXTI3_IRQHandler(void)__attribute__((naked)) __attribute__((optimize("Os")));
void  EXTI3_IRQHandler(void) {

// Interruption ACK

    EXTI->PR |= (1<<BKERNNSWIMSG);

// Recover the swi message
// r0 contains the stack

    __asm__ volatile (
        "cpsid    i          \n"      // 
        "mrs     r0,psp      \n"      // r0 = stack
        "add     r1,r0,#36   \n"      // r1 = msg add (normal stk frame)
        "tst     r14,#0x10   \n"      // 
    );
}
```

```

"it      eq          \n"    //
"addeq   r1,r0,#(36+72) \n"  // r1 = msg add (stk with fpu reg)
"ldr     r1,[r1]       \n"  // r1 = message
"str     r1,%0         \n"  // Save the message on the vMessage
:
: "m"  (vKern_message)
: KSAVEREGISTERS
);

// Save the registers r4..r11 and the basepri
// r0 contains the stack

__asm__ volatile (
"mrs      r1,basepri      \n"  // r1 = basepri
"stmdb   r0!,{r1,r4-r11} \n"  // Save the register list
"tst     r14,#0x10        \n"  //
"it      eq          \n"    //
"vstmdbeq r0!,{s16-s31} \n"  // If used, save the fp registers
"stmdb   r0!,{r14}        \n"  // Save the EXCEPTION return
"msr     psp,r0          \n"  //
"str     r0,%0          \n"  // Save the stack of the old process
"cpsie   i             \n"  //
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

// Change the context due to a message

_newContextMSG();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
"cpsid   i             \n"  //
"ldr     r0,%0          \n"  // Recover the stack of the new process
"ldmia   r0!,{r14}        \n"  // Recover the EXCEPTION return
"tst     r14,#0x10        \n"  //
"it      eq          \n"    //
"vlmdmiaeq r0!,{s16-s31} \n"  // If used, restore the fp registers
"ldmia   r0!,{r1,r4-r11} \n"  // Restore the register list
"msr     psp,r0          \n"  // New stack
"msr     basepri,r1      \n"  // Restore the basepri
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

```

```
// Return

    __asm__ volatile (
        "cpsie      i          \n"    //
        "dmb        \n"    //
        "dsb        \n"    //
        "isb        \n"    //
        "bx         lr         \n"    // Return
    );
}

/*
 * \brief _newContextMSG
 *
 * - Change the context due to a message
 *   - INT acknowledge and new time for the next process
 * - Change the context and prepare the next process
 *   - Stop time of the process
 *   - Clear the interruption
 *   - Disable the timer
 *   - Change the context
 *   - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextMSG(void) {

    _timeStop();
    if ((TIM2->SR & TIM_SR UIF) !=0) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackTrap(vKern_message);
    _timeStart();

#ifdef __WITHSTAT__
    stat_statistic(vKern_backwardProcess,
                  vTimeStart,
                  vTimeStop,
                  vTimeLastStart,
                  vTimeException);
    vTimeException = 0;
#endif
}
```

```
/*
 * \brief _timeStart
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStart(void) {

    vTimeLastStart = vTimeStart;
    vTimeStart      = TIM3->CNT;
}

/*
 * \brief _timeStop
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStop(void) {

    vTimeStop = TIM3->CNT;
}

/*
 * \brief _timeStamp
 *
 * - Maintain a 64-bit timer with 1-us of resolution
 *
 */
static void _timeStamp(void) {
    uint16_t      newTime;
    static uint16_t oldTime = 0;

    INTERRUPTION_OFF;
    newTime = TIM3->CNT;
    if (newTime < oldTime) vTiccount += 0x10000;

    vTiccount = (vTiccount & 0xFFFFFFFFFFFF0000) | newTime;
    oldTime = newTime;
    INTERRUPTION_RESTORED;
}
```

4.6.3. The kernel model for the ARM Cortex M7

```
/*
; model_kernel_tim_1_2_3_4_ex2_ex3.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author:: efr          $: Author of last commit
; $Rev::: 11           $: Revision of last commit
; $Date::: 2016-10-22 21:00:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        Model for controlling the "model_kernel_tim_1_2_3_4_ex2_ex3" device.
;
;             Tim 1 - 1-ms reference time
;             Tim 2 - 20-ms      process timeout
;             Tim 3 - 1-us free running timer
;             Tim 4 - user defined intervals for precise state-machines
;             ex2   - software interruption for preemption
;             ex3   - software interruption for message passing
;
;             This code has to support the following routines:
;             - kernel_init
;             - kernel_runKernel
;             - kernel_loadOverlay
;             - kernel_setLowPower
;             - kernel_runPrecise
;             - kernel_initStackFrame
;             - kernel_setUnixTime
;             - kernel_getUnixTime
;             - kernel_timeStamp
;             - kernel_timeNext
;             - kernel_interruptionFast
;             - kernel_switcher
;             - kernel_message
;             - kernel_getPC
;             - kernel_getFunctionName
;
```

```
; (c) 1992-2017, Franzi Edo.  
;  
;  
; Franzi Edo.          _ _ / / / _ \ _ /  
; 5-Route de Cheseaux   / / / , < / / / / \ _ \  
; CH 1400 Cheseaux-Noréaz / / / / | / / / / _ / /  
;                           \_,/_/ |_\_ / / _ / /  
; edo.franzi@ukos.ch  
  
;  
; This program is free software: you can redistribute it and/or modify  
; it under the terms of the GNU Affero General Public License as published by  
; the Free Software Foundation, either version 3 of the License, or  
; (at your option) any later version.  
  
;  
; This program is distributed in the hope that it will be useful,  
; but WITHOUT ANY WARRANTY; without even the implied warranty of  
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
; GNU Affero General Public License for more details.  
  
;  
; You should have received a copy of the GNU Affero General Public License  
; along with this program. If not, see <http://www.gnu.org/licenses/>.  
;  
;-----  
*/
```

```
#define KFPRET1          1000000
#define KFINTT1          (1/KTTIMEFAST)
#define KPSCT1           ((KFREQUENCY_84/KFPRET1)-1)
#define KARRT1           ((KFPRET1/KFINTT1)-1)

#define KFPRET2          1000000
#define KFINTT2          (1/KTTIMESHARING)
#define KPSCT2           ((KFREQUENCY_84/KFPRET2)-1)
#define KARRT2           ((KFPRET2/KFINTT2)-1)

#define KFPRET3          1000000
#define KFINTT3          KTTIMESTATISTIC
#define KPSCT3           ((KFREQUENCY_84/KFPRET3)-1)
#define KARRT3           (-1)

extern void (*vExce_indIntVectors[KNBINTERRUPTIONS])(void);
extern void (*vKern_codeRoutine)(uint8_t state);
static volatile void (*vCodeRunPrecise)(void);
extern volatile proc_t *vKern_backwardProcess;
static volatile uint16_t vTimeStart = 0;
static volatile uint16_t vTimeStop = 0;
static volatile uint16_t vTimeLastStart = 0;
static volatile uint64_t vTiccount = 0;
static volatile uint32_t vUnixTime;
volatile uint32_t vKern_stackProcess;
volatile uint32_t vKern_message;

// Prototypes
// =====

static void _TIM4_IRQHandler(void);
static void _newContextTOU(void);
static void _newContextMSG(void);
static void _timeStart(void);
static void _timeStop(void);
static void _timeStamp(void);
```

```
/*
 * \brief kernel_init
 *
 * - Turn on the timers
 *
 */
void  kernel_init(void) {

    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;

    vUnixTime = 0;
}

/*
 * \brief kernel_runKernel
 *
 * - Initialize all the timers (1-ms & 20-ms and time-stamp)
 * - Enable the time sharing and all the uKernel interruptions
 *
 */
void  kernel_runKernel(void) {

    NVIC->IP[TIM1_UP_TIM10_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM1_UP_TIM10_IRQn & 0x1F));

    NVIC->IP[TIM2_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM2_IRQn & 0x1F));

    NVIC->IP[TIM4_IRQn] = (KINT_LEVEL_TIMERS<<4);
    NVIC->ISER[0] |= (1<<(TIM4_IRQn & 0x1F));

    NVIC->IP[EXTI2_IRQn] = (KINT_LEVEL_ALL<<4);
    NVIC->ISER[0] |= (1<<(EXTI2_IRQn & 0x1F));

    NVIC->IP[EXTI3_IRQn] = (KINT_LEVEL_SWI<<4);
    NVIC->ISER[0] |= (1<<(EXTI3_IRQn & 0x1F));

// Timer 1 (1-ms)

    TIM1->PSC = KPSCT1;
    TIM1->ARR = KARRT1;
    TIM1->DIER = TIM_DIER_UIE;
    TIM1->CR1 |= TIM_CR1_CEN;
```

```
// Timer 2 (20-ms)

    TIM2->PSC  = KPSCT2;
    TIM2->ARR  = KARRT2;
    TIM2->DIER = TIM_DIER_UIE;
    TIM2->CR1 |= TIM_CR1_CEN;

// Timer 3 (free running timer for statistics)

    TIM3->PSC  = KPSCT3;
    TIM3->ARR  = KARRT3;
    TIM3->DIER = 0;
    TIM3->CR1 |= TIM_CR1_CEN;

// The software interruptions

    EXTI->IMR |= (1<<BKERNPREEMPTION);
    EXTI->IMR |= (1<<BKERNSWIMSG);
}

#endif (defined(__WITHOVLY__))
/*
 * \brief kernel_loadOverlay
 *
 * - Load an overlay process inside the overlay RAM
 *
 */
void  kernel_loadOverlay(uint32_t idModule) {

}
```

```
/*
 * \brief kernel_setLowPower
 *
 * - Set the low power mode
 *
 */
void  kernel_setLowPower(uint8_t mode) {

    switch (mode) {
        case KCPUPNORMAL:
        case KCPUPDEEP:
        case KCPUPHIBERNATE:
        case KCPUPSTOP:
        default: {

            // Waiting for an interruption

            WAITING_INTERRUPT;
            break;
        }
    }
}
```

```
/*
 * \brief kernel_runPrecise
 *
 * - Initialize the runPrecise function
 */
void  kernel_runPrecise(uint16_t time, void (*code)()) {

    vExce_indIntVectors[TIM4_IRQn] = aTIM4_IRQHandler;

    vCodeRunPrecise = code;
    switch (time) {
        case 0: {
            TIM4->DIER = 0;
            TIM4->CR1 &= ~TIM_CR1_CEN;
            break;
        }
        default: {
            TIM4->DIER = 0;
            TIM4->CR1 &= ~TIM_CR1_CEN;
            TIM4->SR  &= ~TIM_SR_UIF;

            TIM4->PSC  = (uint16_t)((KFREQUENCY_APB2/1000000)-1);
            TIM4->ARR  = time - 1;
            TIM4->DIER = TIM_DIER_UIE;
            TIM4->CR1 |= TIM_CR1_CEN;
            break;
        }
    }
}

static void  aTIM4_IRQHandler(void) {
    void  (*code)(void);

    // INT acknowledge

    if (TIM4->SR & TIM_SR_UIF)
        TIM4->SR &= ~TIM_SR_UIF;

    code = (void (*)(void))vCodeRunPrecise;
    code();
}
```

```
/*
 * \brief kernel_initStackFrame
 *
 * - Initialize and prepare the stack frame for a process
 */
void  kernel_initStackFrame(volatile proc_t *handle, const void *argument) {
    volatile      uint32_t      *stackFrame, i;

    // Fill the stack with a "magic" pattern (for statistic)

    stackFrame = handle->oSpecification.oStkSup;                                //
    for (i = 0; i < handle->oSpecification.oSzStk; i++)                         //
        *--stackFrame = (('u'<<24) | ('K'<<16) | ('O'<<8) | 'S');           //

    // Prepare the stack frame

    stackFrame = handle->oSpecification.oStkSup;                                //
    *--stackFrame = 0x01000000;                                                 // xPSR
    *--stackFrame = (uint32_t)handle->oSpecification.oCode;                      // PC
    *--stackFrame = 0xFFFFFFFFFD;                                                // LR
    *--stackFrame = 0x12121212;                                                 // r12
    *--stackFrame = 0x03030303;                                                 // r3
    *--stackFrame = 0x02020202;                                                 // r2
    *--stackFrame = 0x01010101;                                                 // r1
    *--stackFrame = (uint32_t)argument;                                           // r0
    *--stackFrame = 0x11111111;                                                 // r11
    *--stackFrame = 0x10101010;                                                 // r10
    *--stackFrame = 0x09090909;                                                 // r9
    *--stackFrame = 0x08080808;                                                 // r8
    *--stackFrame = 0x07070707;                                                 // r7
    *--stackFrame = 0x06060606;                                                 // r6
    *--stackFrame = 0x05050505;                                                 // r5
    *--stackFrame = 0x04040404;                                                 // r4
    *--stackFrame = (KINT_IMASK_ALL<<4);                                     // basepri
    *--stackFrame = 0xFFFFFFFFFD;                                              // exc ret

    handle->oSpecification.oStkSup = (uint32_t *)stackFrame;
}
```

```
/*
 * \brief kernel_setUnixTime
 *
 * - Set the UNIX absolute time
 *
 */
void  kernel_setUnixTime(uint32_t unixTime) {

    vUnixTime = unixTime;
}

/*
 * \brief kernel_getUnixTime
 *
 * - Get the UNIX absolute time
 *
 */
void  kernel_getUnixTime(uint32_t *unixTime) {

    *unixTime = vUnixTime;
}

/*
 * \brief kernel_timeStamp
 *
 * - Give a new 64-bit time-stamp (1-us of resolution)
 *   - 0x0000000000000000 to 0xFFFFFFFFFFFFFFFFFF incremental counter
 *
 */
void  kernel_timeStamp(uint64_t *timeStamp) {

    _timeStamp();
    *timeStamp = vTiccount;
}

/*
 * \brief kernel_timeNext
 *
 * - Give a new time for a process
 *
 */
void  kernel_timeNext(void) {

    TIM2->CNT  = 0;
    TIM2->ARR  = KARRT2;
    TIM2->SR   &= ~TIM_SR UIF;
    TIM2->CR1 |= TIM_CR1_CEN;
}
```

```
/*
 * \brief kernel_getPC
 *
 * - Get a value of the PC
 *
 */
void  kernel_getPC(const uint32_t *stackProcess, uint32_t *pc) {
    uint8_t          pcOffset = 1;
    uint32_t         lr;

// uKOS stackframe:
// 
// stmdb          r0!,{r14}           -> pcOffset = 1
// 
// if the fpu was used
// vstmdbeq       r0!,{s16-s31}      -> pcOffset += ((31 - 16) + 1)
// 
// stmdb          r0!,{r1,r4-r11}     -> pcOffset += 1 + ((11 - 4) + 1)
// 
// Automatic stacked
// 
//          r3-r0           -> pcOffset += ((3 - 0) + 1)
//          r12             -> pcOffset += 1
//          lr (r14)        -> pcOffset += 1
//          PC

    lr = stackProcess[0];

// If the fpu was used                         s31 to s16
// 
// if ((lr & (1<<4)) == 0) { pcOffset += ((31 - 16) + 1); }

//          r1,          r11 to r4,          r3 to r0,          r12,   r14
//          --          -----          -----          ---   ---
// pcOffset += 1      + ((11 - 4) + 1)      + ((3 - 0) + 1)      + 1      + 1;
*pc = (stackProcess[pcOffset]);
}
```

```
/*
 * \brief kernel_getFunctionName
 *
 * - Get the function name prepended by the gcc compiler
 *   (extraction using the option "-mpoke-function-name"
 *
 * t0
 *     .ascii "arm_poke_function_name", 0
 *     .align
 *
 * t1
 *     .word 0xffff00000 + (t1 - t0)
 *     arm_poke_function_name
 *     mov    ip, sp
 *     stmdfd sp!, {fp, ip, lr, pc}
 *     sub    fp, ip, #4
 *
 */
void  kernel_getFunctionName(const uint32_t pc, const char_t **function) {
    uint16_t      off;
    uint32_t      nameLen;
    const uint32_t      *ptr;

    ptr = (const uint32_t *)((pc + 2) & (~0x3U));

    for (off = 1; (off < 16*1024) && ((uint32_t)&ptr[-off] > 0x0); ++off) {
        if ((ptr[-off] & 0xFFFFFFF00) == 0xFF000000) {
            nameLen = ptr[-off] & 0xFFU;
            *function = &((const char_t*)&ptr[-off])[-nameLen];
            return;
        }
    }
    *function = NULL;
}
```

```
// Local routines
// =====

/*
 * \brief TIM1_UP_TIM10_IRQHandler
 *
 * - Fine time interruption
 * - Increment the tic-count 1-us variable
 * - Verify the timeout condition of the suspended processes
 *
 */
void    TIM1_UP_TIM10_IRQHandler(void) {
    static uint32_t      counter = 0;

// INT acknowledge
// Tic_count++ and verify the timeout conditions
// Increment the UNIX time seconds

    if ((TIM1->SR & TIM_SR UIF) !=0) {
        TIM1->SR &= ~TIM_SR UIF;
    }

    scheCallBackFast();
    if (counter++ % (1000/KTIMEUNIT) == 0) {
        vUnixTime++;
    }
    _timeStamp();
}
```

```
/*
 * \brief EXTI2_IRQHandler
 *
 * - Preemption
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *
 *          Normal stack frame      Stack frame with fpu
 *          IOFF                  IOFF
 *
 *          ->                   FPSCR
 *          ->                   S15..S0
 *          ->      xPSR           xPSR
 *          ->      PC              PC
 *          ->      LR(R14)        LR(R14)
 *          ->      R12             R12
 *          ->      R3..R0         R3..R0      Block stacked by an exception
 *
 *          ->      R11..R4        R11..R4
 *          ->      BASEPRI        BASEPRI
 *          ->                   S31..S16
 *          ->      LR(R14)        LR(R14)      Block stacked manually
 *
 *
 * !!! Do not generate prologue/epilogue sequences
 */
// Test for GCC version for adapting the assembler context code

#if      (uKOS_GCC_VERSION > 40805)
#define      KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define      KSAVEREGISTERS
#endif

void  EXTI2_IRQHandler(void) __attribute__ ((naked)) __attribute__ ((optimize("Os")));
void  EXTI2_IRQHandler(void) {

// Interruption ACK

    EXTI->PR |= (1<<BKERNPREEMPTION);

// Save the registers r4..r11 and the basepri
// r0 contains the stack
```

```

__asm__ volatile (
    "cpsid      i          \n"      //
    "mrs       r0,psp      \n"      // r0 = stack
    "mrs       r1,basepri  \n"      // r1 = basepri
    "stmdb     r0!,{r1,r4-r11} \n"   // Save the register list
    "tst       r14,#0x10   \n"      //
    "it        eq         \n"      //
    "vstmdbeq  r0!,{s16-s31} \n"   // If used, save the fp registers
    "stmdb     r0!,{r14}    \n"      // Save the EXCEPTION return
    "msr       psp,r0     \n"      //
    "str       r0,%0      \n"      // Save the stack of the old process
    "cpsie    i          \n"      //
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Change the context due to a timeout

_newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
    "cpsid      i          \n"      //
    "ldr       r0,%0      \n"      // Recover the stack of the new process
    "ldmia    r0!,{r14}    \n"      // Recover the EXCEPTION return
    "tst       r14,#0x10   \n"      //
    "it        eq         \n"      //
    "vldmiaeq  r0!,{s16-s31} \n"   // If used, restore the fp registers
    "ldmia    r0!,{r1,r4-r11} \n"   // Restore the register list
    "msr       psp,r0     \n"      // New stack
    "msr       basepri,r1  \n"      // Restore the basepri
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Return

__asm__ volatile (
    "cpsie    i          \n"      //
    "dmb      \n"      //
    "dsb      \n"      //
    "isb      \n"      //
    "bx       lr        \n"      // Return
);
}

```

```

/*
 * \brief TIM2_IRQHandler
 *
 * - Time slice timeout
 * - Save the context
 * - Save the context stack
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
 *

*           Normal stack frame          Stack frame with fpu
*           IOFF                      IOFF
*
*           ->                      FPSCR
*           ->                      S15..S0
*           ->      xPSR            xPSR
*           ->      PC              PC
*           ->      LR(R14)        LR(R14)
*           ->      R12             R12
*           ->      R3..R0         R3..R0      Block stacked by an exception
*
*           ->      R11..R4         R11..R4
*           ->      BASEPRI        BASEPRI
*           ->                      S31..S16
*           ->      LR(R14)        LR(R14)      Block stacked manually
*
* !!! Do not generate prologue/epilogue sequences
*/
// Test for GCC version for adapting the assembler context code

#if   (uKOS_GCC_VERSION > 40805)
#define     KSAVEREGISTERS      "r4", "r5", "r6", "r7", "r8", "r9", "r10", "r11"
#else
#define     KSAVEREGISTERS
#endif
void  TIM2_IRQHandler(void) __attribute__ ((naked)) __attribute__ ((optimize("Os")));
void  TIM2_IRQHandler(void) {

// Save the registers r4..r11 and the basepri
// r0 contains the stack

    __asm__ volatile (
        "cpsid    i                  \n"    //
        "mrs     r0,psp             \n"    // r0 = stack
        "mrs     r1,basepri         \n"    // r1 = basepri
        "stmdb   r0!,{r1,r4-r11}    \n"    // Save the register list
}

```

```

    "tst      r14,#0x10          \n"    //
    "it       eq                 \n"    //
    "vstmdbeq r0!,{s16-s31}     \n"    // If used, save the fp registers
    "stmdb   r0!,{r14}          \n"    // Save the EXCEPTION return
    "msr     psp,r0            \n"    //
    "str     r0,%0              \n"    // Save the stack of the old process
    "cpsie   i                 \n"    //
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Change the context due to a timeout

    _newContextTOU();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
    "cpsid   i                  \n"    //
    "ldr     r0,%0              \n"    // Recover the stack of the new process
    "ldmia   r0!,{r14}          \n"    // Recover the EXCEPTION return
    "tst     r14,#0x10          \n"    //
    "it      eq                 \n"    //
    "vldmiaeq r0!,{s16-s31}     \n"    // If used, restore the fp registers
    "ldmia   r0!,{r1,r4-r11}    \n"    // Restore the register list
    "msr     psp,r0            \n"    // New stack
    "msr     basepri,r1        \n"    // Restore the basepri
    :
    : "m"  (vKern_stackProcess)
    : KSAVEREGISTERS
    );

// Return

__asm__ volatile (
    "cpsie   i                  \n"    //
    "dmb    "
    "dsb    "
    "isb    "
    "bx     lr                 \n"    // Return
    );
}

```

```
/*
 * \brief _newContextTOU
 *
 * - Change the context due to a timeout
 *   - INT acknowledge and new time for the next process
 *   - Change the context and prepare the next process
 *     - Stop time of the process
 *     - Clear the interruption
 *     - Disable the timer
 *     - Change the context
 *     - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextTOU(void) {

    _timeStop();
    if ((TIM2->SR & TIM_SR UIF) !=0) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackSlow();
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess,
                   vTimeStart,
                   vTimeStop,
                   vTimeLastStart,
                   vTimeException);
    vTimeException = 0;
#endif
}
```

```

/*
 * \brief EXTI3_IRQHandler
 *
 * - TRAP
 * - Save the context
 * - Save the context stack
 * - Process the message sent
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 * - Save the full stack frame (uKOS like)
*
*      Normal stack frame          Stack frame with fpu
*      IOFF                      IOFF
*
*      ->    Message        +36   +108
*      ->    Message        +32   +104
*
*      ->                      FPSCR
*      ->                      S15..S0
*      ->    xPSR           xPSR
*      ->    PC              PC
*      ->    LR(R14)         LR(R14)
*      ->    R12             R12
*      ->    R3..R0          R3..R0      Block stacked by an exception
*
*      ->    R11..R4          R11..R4
*      ->    BASEPRI         BASEPRI
*      ->                      S31..S16
*      ->    LR(R14)         LR(R14)      Block stacked manually
*
* !!! Do not generate prologue/epilogue sequences
*
*/
void  EXTI3_IRQHandler(void)__attribute__((naked)) __attribute__((optimize("Os")));
void  EXTI3_IRQHandler(void) {

// Interruption ACK

    EXTI->PR |= (1<<BKERNNSWIMSG);

// Recover the swi message
// r0 contains the stack

    __asm__ volatile (
        "cpsid    i                  \n"    //
        "mrs     r0,psp            \n"    // r0 = stack
        "add     r1,r0,#36          \n"    // r1 = msg add (normal stk frame)
        "tst     r14,#0x10          \n"    //
    );
}

```

```

"it      eq          \n"    //
"addeq   r1,r0,#(36+72) \n"  // r1 = msg add (stk with fpu reg)
"ldr     r1,[r1]       \n"  // r1 = message
"str     r1,%0         \n"  // Save the message on the vMessage
:
: "m"  (vKern_message)
: KSAVEREGISTERS
);

// Save the registers r4..r11 and the basepri
// r0 contains the stack

__asm__ volatile (
"mrs      r1,basepri      \n"  // r1 = basepri
"stmdb   r0!,{r1,r4-r11} \n"  // Save the register list
"tst     r14,#0x10        \n"  //
"it      eq          \n"    //
"vstmdbeq r0!,{s16-s31} \n"  // If used, save the fp registers
"stmdb   r0!,{r14}        \n"  // Save the EXCEPTION return
"msr     psp,r0          \n"  //
"str     r0,%0          \n"  // Save the stack of the old process
"cpsie   i             \n"  //
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

// Change the context due to a message

_newContextMSG();

// Restore the registers r4..r11 and the basepri

__asm__ volatile (
"cpsid   i             \n"  //
"ldr     r0,%0          \n"  // Recover the stack of the new process
"ldmia   r0!,{r14}        \n"  // Recover the EXCEPTION return
"tst     r14,#0x10        \n"  //
"it      eq          \n"    //
"vlmdmiaeq r0!,{s16-s31} \n"  // If used, restore the fp registers
"ldmia   r0!,{r1,r4-r11} \n"  // Restore the register list
"msr     psp,r0          \n"  // New stack
"msr     basepri,r1      \n"  // Restore the basepri
:
: "m"  (vKern_stackProcess)
: KSAVEREGISTERS
);

```

```
// Return

    __asm__ volatile (
        "cpsie      i          \n"    //
        "dmb        \n"    //
        "dsb        \n"    //
        "isb        \n"    //
        "bx         lr         \n"    // Return
    );
}

/*
 * \brief _newContextMSG
 *
 * - Change the context due to a message
 *   - INT acknowledge and new time for the next process
 * - Change the context and prepare the next process
 *   - Stop time of the process
 *   - Clear the interruption
 *   - Disable the timer
 *   - Change the context
 *   - Start time of the process
 *
 */
static void __attribute__((noinline))_newContextMSG(void) {

    _timeStop();
    if ((TIM2->SR & TIM_SR UIF) !=0) {
        TIM2->SR &= ~TIM_SR UIF;
    }
    TIM2->CR1 &= ~TIM_CR1_CEN;

    scheCallBackTrap(vKern_message);
    _timeStart();

#ifdef __WITHSTAT__
    stat_statistic(vKern_backwardProcess,
                  vTimeStart,
                  vTimeStop,
                  vTimeLastStart,
                  vTimeException);
    vTimeException = 0;
#endif
}
```

```
/*
 * \brief _timeStart
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStart(void) {

    INTERRUPTION_OFF;
    vTimeLastStart = vTimeStart;
    vTimeStart      = TIM3->CNT;
    INTERRUPTION_RESTORED;
}

/*
 * \brief _timeStop
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStop(void) {

    vTimeStop = TIM3->CNT;
}

/*
 * \brief _timeStamp
 *
 * - Maintain a 64-bit timer with 1-us of resolution
 *
 */
static void _timeStamp(void) {
    uint16_t      newTime;
    static uint16_t oldTime = 0;

    INTERRUPTION_OFF;
    newTime = TIM3->CNT;
    if (newTime < oldTime) vTiccount += 0x10000;

    vTiccount = (vTiccount & 0xFFFFFFFFFFFF0000) | newTime;
    oldTime = newTime;
    INTERRUPTION_RESTORED;
}
```

4.6.4. The kernel model for the IcyCAM (icyflex-1)

```
/*
;   model_kernel_tim_0_1_2_tp1.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author:: efr           $: Author of last commit
; $Rev::: 11              $: Revision of last commit
; $Date::: 2016-10-22 21:00:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        Model for controlling the "model_kernel_tim_0_1_2_tp1" device.
;
;               Tim 0 - 1-ms reference time
;               Tim 1 - 20-ms      process timeout
;               Tim 2 - 1-us free running timer
;               tp1    - software interruption for message passing
;
; This code has to support the following routines:
;       - kernel_init
;       - kernel_runKernel
;       - kernel_loadOverlay
;       - kernel_setLowPower
;       - kernel_runPrecise
;       - kernel_initStackFrame
;       - kernel_setUnixTime
;       - kernel_getUnixTime
;       - kernel_timeStamp
;       - kernel_timeNext
;       - kernel_interruptionFast
;       - kernel_switcher
;       - kernel_message
;       - kernel_getPC
;       - kernel_getFunctionName
;
```

```
; (c) 1992-2017, Franzi Edo.  
;  
;  
; Franzi Edo.          _ _ / / / _ \ _ /  
; 5-Route de Cheseaux   / / / , < / / / / \ _ \  
; CH 1400 Cheseaux-Noréaz / / / / | / / / / _ / /  
;                           \_,/_/ |_\_ / / _ / /  
; edo.franzi@ukos.ch  
  
;  
; This program is free software: you can redistribute it and/or modify  
; it under the terms of the GNU Affero General Public License as published by  
; the Free Software Foundation, either version 3 of the License, or  
; (at your option) any later version.  
  
;  
; This program is distributed in the hope that it will be useful,  
; but WITHOUT ANY WARRANTY; without even the implied warranty of  
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
; GNU Affero General Public License for more details.  
  
;  
; You should have received a copy of the GNU Affero General Public License  
; along with this program. If not, see <http://www.gnu.org/licenses/>.  
;  
;-----  
*/
```

```
#define KDIV32 5
#define KTTIMEFAST ((KTIMEUNIT*KFREQUENCY_33/1000) + 1)
#define KTTIMESHARING ((KTIMEPROC*KFREQUENCY_33/1000) + 1)
#define KTTIMESTATISTIC -1

extern void (*vKern_codeRoutine)(uint8_t state);
extern volatile proc_t *vKern_backwardProcess;
static volatile uint16_t vTimeStart = 0;
static volatile uint16_t vTimeStop = 0;
static volatile uint16_t vTimeLastStart = 0;
static uint32_t vUnixTime;
static uint64_t vTiccount = 0;
volatile uint32_t vKern_stackProcess;
volatile uint32_t vKern_message;

// Prototypes
// =====

extern void handle_kern_switcher(void);
extern void handle_kern_message(void);
extern void handle_kern_interruptionFast(void);
static void _timeStart(void);
static void _timeStop(void);
static void _timeStamp(void);
static void _loadTimer(uint16_t *timer);
```

```
/*
 * \brief kernel_init
 *
 * - Initialize the UNIX time 1-1-1970 0h 0m 0s
 * - Initialize the interruption vectors
 *
 */
void  kernel_init(void) {

    vUnixTime = 0;
    EXCEPTION_VECTOR(KTRAP1EXCEPTION, handle_kernel_message);
}

/*
 * \brief kernel_runKernel
 *
 * - Initialize all the timers (1-ms & 20-ms and time-stamp)
 * - Enable the time sharing and all the uKernel interruptions
 *
 */
void  kernel_runKernel(void) {

// Timer 0 (1-ms)

    *TIMERO_PWM_LO = KTTIMEFAST;
    *TIMERO_IRQEN  = (1<<BBOUND_REACHED);

// Timer 1 (20-ms)

    *TIMER1_PWM_LO = KTTIMESHARING;
    *TIMER1_IRQEN  = (1<<BBOUND_REACHED);

// Timer 2 (free running timer for statistics)

    *TIMER2_PWM_LO = KTTIMESTATISTIC;

// Interrupt controller & enable the timers

    *RQCTRL IRQ_VEC_5 = (uint32_t)(handle_kernel_interruptionFast);
    *RQCTRL IRQ_VEC_6 = (uint32_t)(handle_kernel_switcher);
    *RQCTRL_SOURCE_EN_W1S = (1<<BTIMER1) | (1<<BTIMER0);
    *RQCTRL_IRQ_CONFIG_0 |= (KINT_LEVEL_TIMERS<<BPRIREQ06) \
                           | (KINT_LEVEL_TIMERS<<BPRIREQ05);
    *RQCTRL_IRQ_LEVEL_EN = KINT_SYS_LEVEL;

    *TIMERO_CTRL = CPT_INTERNAL | (1<<BTIMER_IRQ_EN) | (1<<BTIMER_EN);
    *TIMER1_CTRL = CPT_INTERNAL | (1<<BTIMER_IRQ_EN) | (1<<BTIMER_EN);
    *TIMER2_CTRL = CPT_INTERNAL | (1<<BTIMER_EN);
}
```

```
#if (defined(__WITHOVLY__))
/*
 * \brief kernel_loadOverlay
 *
 * - Load an overlay process inside the overlay RAM
 *
 */
void  kernel_loadOverlay(uint32_t idModule) {

}

#endif

/*
 * \brief kernel_setLowPower
 *
 * - Set the low power mode
 *
 */
void  kernel_setLowPower(uint8_t mode) {

    cb_setLowPower(mode);
}

/*
/*
 * \brief kernel_runPrecise
 *
 * - Initialize the runPrecise function
 *
 */
void  kernel_runPrecise(uint16_t time, void (*code)()) {

}
```

```

/*
 * \brief kernel_initStackFrame
 *
 * - Initialize and prepare the stack frame for a process
 */
void  kernel_initStackFrame(volatile proc_t *handle, const void *argument) {
    uint32_t      *stackFrame, add, addMsb, addLsb, i;

//  Return address format inside the hardware stack
//
//  --          16 15   9   8          4          3 2   10
//  -- Word is: ... | Return Address | ..... | Exception Status | pcm | 00 (jsr)
//  --       ... | Return Address | ..... | Exception Status | pcm | 01 (exc)
//  --       lend | l beg     |           | lrit          | pcm | 10 (loop)
//  --       lend | l beg     |           | lrit          | pcm | 11 (rep)
//           |           |           |           |           |
//           |           |           |           | 1   1001    | 00  | 01
//  MSB word: Bits 07..00      Address MSB
//  LSB word: Bits 31..16      Address LSB
//  LSB word: Bits 15..09      -
//  LSB word: Bits 08..04      0x19 (No Exceptions)
//  LSB word: Bits 01..00      0x01 (Exceptions)

    add = (uint32_t)handle->oSpecification.oCode;           //
    add = add>>2;                                         // Address / 4 (long w)
    addMsb = (add>>16) & 0x000000FF;                      //
    addLsb = (add<<16) & 0xFFFFF000;                      //
    addLsb = addLsb | 0x0191;                                //

// Fill the stack with a "magic" pattern (for statistic)

    stackFrame = handle->oSpecification.oStkSup;
    for (i = 0; i < handle->oSpecification.oSzStk; i++)
        *--stackFrame = (('u'<<24) | ('K'<<16) | ('O'<<8) | 'S');

// Prepare the stack frame

    stackFrame = handle->oSpecification.oStkSup;           //
    *--stackFrame = 0;                                       // r7
    *--stackFrame = 0;                                       // r6
    *--stackFrame = 0;                                       // r5
    *--stackFrame = 0;                                       // r4
    *--stackFrame = 0;                                       // r3
    *--stackFrame = 0;                                       // r2
    *--stackFrame = 0;                                       // r1
    *--stackFrame = 0;                                       // r0
    *--stackFrame = 0;                                       // px6
    *--stackFrame = 0;                                       // px5
    *--stackFrame = 0;                                       // px4

```

```
*--stackFrame = 0;                                // px3
*--stackFrame = 0;                                // px2
*--stackFrame = 0;                                // px1
*--stackFrame = (uint32_t)argument;                // px0
*--stackFrame = 0x00000000;                         // psu1 L (PB PA)
*--stackFrame = 0x00000000;                         // psu1 H (DM PF)
*--stackFrame = 0x00002000;                         // psu0 L (HD EC)
*--stackFrame = 0x00008000;                         // psu0 H (EX IN)
*--stackFrame = KINT_IMASK_ALL;                    // iel
*--stackFrame = 0;                                // sbr
*--stackFrame = addLsb;                            // address LSB
*--stackFrame = addMsb;                            // address MSB
*--stackFrame = 1;                                // scnt

    handle->oSpecification.oStkSup = (uint32_t *)stackFrame;
}

/*
 * \brief kernel_setUnixTime
 *
 * - Set the UNIX absolute time
 *
 */
void    kernel_setUnixTime(uint32_t unixTime) {

    vUnixTime = unixTime;
}

/*
 * \brief kernel_getUnixTime
 *
 * - Get the UNIX absolute time
 *
 */
void    kernel_getUnixTime(uint32_t *unixTime) {

    *unixTime = vUnixTime;
}
```

```
/*
 * \brief kernel_timeStamp
 *
 * - Give a new 64-bit time-stamp (1-us of resolution)
 *   - 0x0000000000000000 to 0xFFFFFFFFFFFFFF incremental counter
 *
 */
void  kernel_timeStamp(uint64_t *timeStamp) {

    _timeStamp();
    *timeStamp = vTiccount;
}

/*
 * \brief kernel_timeNext
 *
 * - Give a new time for a process
 *
 */
void  kernel_timeNext(void) {

    *TIMER1_PWM_LO = KTTIMESHARING;
    *TIMER1_CTRL    = CPT_INTERNAL | (1<<BTIMER_IRQ_EN) | (1<<BTIMER_EN);
}

/*
 * \brief kernel_getPC
 *
 * - Get a value of the PC
 *
 */
void  kernel_getPC(const uint32_t *stackProcess, uint32_t *pc) {

    *pc = 0;
}

/*
 * \brief kernel_getFunctionName
 *
 * - Get the function name prepended by the gcc compiler
 *   (extraction using the option "-mpoke-function-name"
 *
 */
void  kernel_getFunctionName(const uint32_t pc, const char_t **function) {

    *function = NULL;
}
```

```
// Local routines
// =====

/*
 * \brief kernel_interruptionFast
 *
 * - Fine time interruption
 * - Increment the tic-count 1-us variable
 * - Verify the timeout condition of the suspended processes
 *
 */
void  kernel_interruptionFast(void) {

// INT acknowledge
// Tic_count++ and verify the timeout conditions
// Increment the UNIX time seconds

    *TIMER0_STATUS = (1<<BBOUND_REACHED);
    sche_callBackFast();
    if (vKern_ticcount % (1000/KTIMEUNIT) == 0) vUnixTime++;
    _timeStamp();
}

/*
 * \brief kernel_switcher
 *
 * - Time slice timeout
 * - Save & change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 */
void  kernel_switcher(void) {

// INT acknowledge and new time for the next process
// Change the context and prepare the next process
// Increment the UNIX time seconds

    _timeStop();
    *TIMER1_STATUS = (1<<BBOUND_REACHED);
    *TIMER1_CTRL    = 0;

    sche_callBackSlow();
    _timeStart();

    #if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess, \
                  vTimeStart, vTimeStop, vTimeLastStart);
    #endif
}
```

```
/*
 * \brief kernel_message
 *
 * - TRAP
 * - Save the context
 * - Save the context stack
 * - Process the message sent
 * - Change the context
 * - Give another timeout to the process
 * - Recover another context
 *
 */
void  kernel_message(void) {

// INT acknowledge and new time for the next process
// Change the context and prepare the next process

    _timeStop();
    *TIMER1_STATUS = (1<<BBOUND_REACHED);
    *TIMER1_CTRL   = 0;

    scheCallBackTrap(vKern_message);
    _timeStart();

#if (defined(__WITHSTAT__))
    stat_statistic(vKern_backwardProcess, \
                   vTimeStart, vTimeStop, vTimeLastStart);
#endif
}

/*
 * \brief _loadTimer
 *
 * - The timer is clocked at 33-MHz
 * So, the timer has to be divided by 33!
 * value/33 ~= value/32 - value/1024 - value/33792 - ...
 *
 * - Approximated at
 * value/33 ~= value/32 - value/1024
 *
 */
static void  _loadTimer(uint16_t *timer) {
    uint32_t      counter, p32, p1024;

    counter = -(*TIMER2_COUNTER);
    p32     = (counter>>KDIV32);
    p1024   = (p32>>KDIV32);
    *timer = ((uint16_t)(p32 - p1024));
}
```

```
/*
 * \brief _timeStart
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStart(void) {

    vTimeLastStart = vTimeStart;
    _loadTimer((uint16_t *)&vTimeStart);
}

/*
 * \brief _timeStop
 *
 * - Give a new time-stamp (1-us of resolution)
 *   - 0x0000 to 0xFFFF incremental counter
 *
 */
static void _timeStop(void) {

    _loadTimer((uint16_t *)&vTimeStop);
}

/*
 * \brief _timeStamp
 *
 * - Maintain a 64-bit timer with 1-us of resolution
 *
 */
static void _timeStamp(void) {
    uint16_t newTime;
    static uint16_t oldTime = 0;

    INTERRUPTION_OFF;
    _loadTimer(&newTime);
    if (newTime < oldTime) vTiccount += 0x10000;

    vTiccount = (vTiccount & 0xFFFFFFFFFFFF0000) | newTime;
    oldTime = newTime;
    INTERRUPTION_RESTORED;
}
```

```
/*
; interruptions.
; =====

;-----+
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
; Version:     1.0
;
; CVS:
; $Author::: efr           $: Author of last commit
; $Rev::: 63              $: Revision of last commit
; $Date::: 2011-06-02 11:10:24$: Date of last commit
;
; Project:     uKOS
; Goal:        Interruption management for the module.
;
; Mode:        This module works in the supervisor space
;
;
;          +-----+
; Message switch |      message      | .32
;          +-----+
;
;
;          +-----+
; Normal switch |      r7..r0       | 8 x .32
;          +-----+
;          |      px6..px0      | 7 x .32
;          +-----+
;          |      psu1          | .64
;          +-----+
;          |      psu0          | .64
;          +-----+
;          |      iel           | .32
;          +-----+
;          |      sbr           | .32
;          +-----+
;          |      scnt          | .32
;          +-----+
;          |      MSB LSB Hard Stack | scnt x .64
;          +-----+
;
; (c) 1992-2017, Franzi Edo.
; -----
;
; Franzi Edo.
; 5-Route de Cheseaux
; CH 1400 Cheseaux-Noréaz
; edo.franzi@ukos.ch
```



```
;  
; This program is free software: you can redistribute it and/or modify  
; it under the terms of the GNU Affero General Public License as published by  
; the Free Software Foundation, either version 3 of the License, or  
; (at your option) any later version.  
;  
; This program is distributed in the hope that it will be useful,  
; but WITHOUT ANY WARRANTY; without even the implied warranty of  
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
; GNU Affero General Public License for more details.  
;  
; You should have received a copy of the GNU Affero General Public License  
; along with this program. If not, see <http://www.gnu.org/licenses/>.  
;  
-----  
*/  
  
.global      _handle_kernel_interruptionFast  
.global      _handle_kernel_switcher  
.global      _handle_kernel_message  
.extern      _kernel_interruptionFast  
.extern      _kernel_switcher  
.extern      _kernel_message  
.extern      _kernel_timeStrt  
.extern      _kernel_timeStop  
.extern      _vStckProcess  
.extern      _vMessage  
.extern      _saveScratchRegisters  
.extern      _restoreScratchRegisters  
  
.section     .text  
  
; Interruption Fast  
;  
-----  
  
_handle_kernel_interruptionFast:  
    jsr          _saveScratchRegisters           ; Save the scratch registers  
    nop  
    sjsr        _kernel_interruptionFast         ; Process the interruption  
    nop  
    jsr          _restoreScratchRegisters        ; Restore the scratch registers  
    nop  
    rte  
    nop
```

```

; Save and restore the context
; -----
;_handle_kernel_switcher:
    mmovl      -(px7),r0-r7          ; r7 .. r0
    mmovl      -(px7),px0-px6        ; px6 .. px0
    pmov       r3:r2,psul          ; PB PA DM PF
    pmov       r1:r0,psu0          ; HD EC EX IN
    mmovl      -(px7),r0-r3          ;
    movil.h    r0.u,#0             ;
    pmov       r0.l,iel            ;
    movbp     +(px7,-4),r0          ; 0x00iel
    movil.h    r0.l,#0             ;
    pmov       iel,r0.l           ; ioff
    pmov       r0,sbr              ;
    movbp     +(px7,-4),r0          ; sbr

    sjsr      _kernel_timeStop      ; Stop time of the process
    nop                   ; ;

    movuiip   r0,#0               ;
    pmov       r0.l,scnt            ;
    addi      r0,#-1              ;
    jmp       pa2,1f              ;
    nop                   ; ;

    loop      r0.l,1f              ;
    pop       r3:r2              ; 64-bits HS
    movbp     +(px7,-4),r2          ; LSB
    movbp     +(px7,-4),r3          ; MSB
1:   addil    r0,#1              ;
    movbp     +(px7,-4),r0          ; scnt

    movil.h    px0.u,#sval.u(_vStckProcess)  ;
    movil.h    px0.l,#sval.l(_vStckProcess)  ;
    movbl     (px0,0),px7          ; Save the pointer

; Call the nano kernel for switching a task
    sjsr      _kernel_switcher      ; Call the nano kernel
    nop                   ; ;

; Restore the new context
    sjsr      _kernel_timeStrt      ; Start time of the process
    nop                   ; ;

    movil.h    px0.u,#sval.u(_vStckProcess)  ;
    movil.h    px0.l,#sval.l(_vStckProcess)  ;

```

```

movbl      px7,(px0,0)          ; Recover the pointer
movbp      r0,(px7,4)+          ; scnt
addil      r0,#-1              ;
jmp       pa2,1f               ;
nop

loop       r0.1,1f              ;
movbp      r3,(px7,4)+          ; MSB
movbp      r2,(px7,4)+          ; LSB
push       r3:r2               ; 64-bits HS

1:   movbp      r0,(px7,4)+          ; sbr
pmov       sbr,r0              ;
movbp      r0,(px7,4)+          ; 0x00iel
pmov       iel,r0.1             ;

mmovl     r0-r3,(px7)+          ;
pmov       psu0,r1:r0            ; HD EC EX IN
pmov       psu1,r3:r2            ; PB PA DM PF
mmovl     px0-px6,(px7)+          ; px6 .. px0
mmovl     r0-r7,(px7)+          ; r7 .. r0
rte
nop

; Save and restore the context (for a trap message)
; -----
.set    posMsg, 84

_handle_kernel_message:
mmovl     -(px7),r0-r7          ; r7 .. r0
mmovl     -(px7),px0-px6          ; px6 .. px0
pmov      r3:r2,psu1            ; PB PA DM PF
pmov      r1:r0,psu0            ; HD EC EX IN
mmovl     -(px7),r0-r3          ; 

movil.h   r0.u,#0              ;
pmov       r0.l,iel              ;
movbp      +(px7,-4),r0            ; 0x00iel
movil.h   r0.l,#0              ;
pmov       iel,r0.1              ; ioff
pmov       r0,sbr                ;
movbp      +(px7,-4),r0            ; sbr

sjsr      _kernel_timeStop        ; Stop time of the process
nop

movil.h   px0.u,#sval.u(_vMessage)  ;
movil.h   px0.l,#sval.l(_vMessage)  ;
movbl      r7,(px7,posMsg)         ; 84 = sum(registers)

```

```

movbl      (px0,0),r7          ; Save the message

movuip     r0,#0              ;
pmov       r0.l,scnt          ;
addi      r0,#-1              ;
jmp       pa2,1f              ;
nop                  ;

loop      r0.l,1f              ;
pop       r3:r2              ; 64-bits HS
movbp     +(px7,-4),r2          ; LSB
movbp     +(px7,-4),r3          ; MSB
1:       addil    r0,#1              ;
movbp     +(px7,-4),r0          ; scnt

movil.h    px0.u,#sval.u(_vStckProcess)   ;
movil.h    px0.l,#sval.l(_vStckProcess)   ;
movbl      (px0,0),px7          ; Save the pointer

; Call the nano kernel for switching a task and
; for the message analisys

sjsr      _kernel_message        ; Call the nano kernel
nop                  ; ;

; Restore the new context

sjsr      _kernel_timeStrt      ; Start time of the process
nop                  ; ;

movil.h    px0.u,#sval.u(_vStckProcess)   ;
movil.h    px0.l,#sval.l(_vStckProcess)   ;

movbl      px7,(px0,0)           ; Recover the pointer
movbp     r0,(px7,4)+          ; scnt
addil    r0,#-1              ;
jmp       pa2,1f              ;
nop                  ;
loop      r0.l,1f              ;
movbp     r3,(px7,4)+          ; MSB
movbp     r2,(px7,4)+          ; LSB
push     r3:r2              ; 64-bits HS

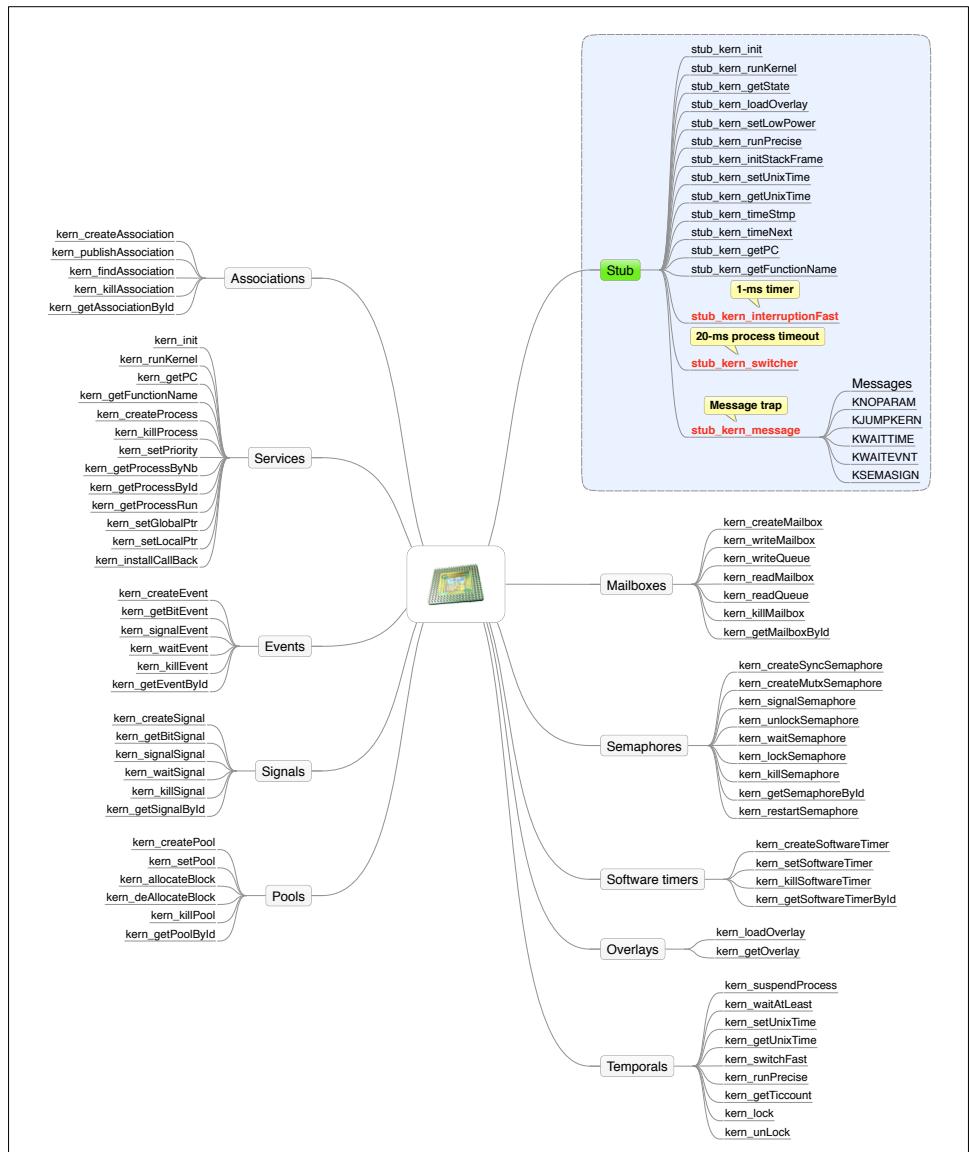
1:       movbp     r0,(px7,4)+          ; sbr
pmov      sbr,r0              ;
movbp     r0,(px7,4)+          ; 0x00iel
pmov      iel,r0.l             ;

mmovl    r0-r3,(px7)+          ;
pmov      psu0,r1:r0            ; HD EC EX IN

```

```
pmov      psu1,r3:r2          ; PB PA DM PF
mmovl    px0-px6,(px7)+       ; px6 .. px0
mmovl    r0-r7,(px7)+       ; r7 .. r0
rte
nop
```

4.7. Groups of functions of the μKernel (families of system calls)



From the point of view of the programmer the μKernel is divided in groups of functions. Seven groups of functions (families of system calls) are used for managing the μKernel as well as the applications; the **service**, the **temporal**, the **semaphore**, the **mailbox**, the **association**, and the **hardware / software event**.

Service System Calls	
kern_init	Initialize the “kern” manager
kern_getState	Get the state of the μKernel
kern_runKernel	Start the multi-process
kern_createProcess	Create and install a process
kern_killProcess	Kill a process
kern_setPriority	Set the a new static priority for a process
kern_getProcessByNb	Get the process by its number
kern_getProcessById	Get the process by its Id
kern_getProcessRun	Get the process of the running one
kern_setGlobalPtr	Set a global pointer
kern_setLocalPtr	Set a local pointer
kern_installCallBack	Install a callback routine
kern_getPC	Get the process PC
kern_getFunctionName	Get the name of a function
Temporal System Calls	
kern_suspendProcess	Suspend the process for a time
kern_waitAtLeast	Wait for short times (at least ... a certain time)
kern_setUnixTime	Set the 32-bit UNIX time
kern_getUnixTime	Get the 32-bit UNIX time
kern_switchFast	Force the context switching
kern_getTiccount	Get the “tic-count” from the start of the μKernel
kern_runPrecise	Cyclically run a function with a precise time (1-μs resolution)

Semaphore System Calls	
kern_createSyncSemaphore	Create a synchro semaphore
kern_createMutexSemaphore	Create a mutex semaphore
kern_signalSemaphore	Signal an event
kern_unlockSemaphore	Unlock the semaphore
kern_waitSemaphore	Waiting for an event
kern_lockSemaphore	Lock the semaphore
kern_killSemaphore	Kill a semaphore
kern_getSemaphoreByld	Get the semaphore by its Id

Mailbox System Calls	
kern_createMailbox	Create a mailbox
kern_writeMailbox	Write a message in the mailbox
kern_writeQueue	Write a message in the queue
kern_readMailbox	Read a message from the mailbox
kern_readQueue	Read a message from the queue
kern_killMailbox	Kill a mailbox
kern_getMailboxByld	Get the mailbox by its Id

Association System Calls	
kern_createAssociation	Create an association
kern_publishAssociation	Publish an association
kern_findAssociation	Find an association
kern_killAssociation	Kill an association
kern_getAssociationByld	Get the association by its Id

Event System Calls	
kern_createEvent	Create an event
kern_getBitEvent	Get the bit that corresponds to an event
kern_signalEvent	Signal an event
kern_waitEvent	Waiting for an event
kern_killEvent	Kill an event
kern_getEventByld	Get the event by its Id

Signal System Calls	
kern_createSignal	Create a signal
kern_getBitSignal	Get the bit that the corresponds to an signal
kern_signalSignal	Signal a signal
kern_waitSignal	Waiting for a signal
kern_killSignal	Kill a signal
kern_getSignalByld	Get the signal by its Id

Software timer System Calls	
kern_createSoftwareTimer	Create a software timer
kern_setSoftwareTimer	Set a software timer
kern_killSoftwareTimer	Kill a software timer
kern_getSoftwareTimerByld	Get the software timer by its Id

Overlay System Calls	
kern_loadOverlay	Load an overlay process inside the overlay RAM segment
kern_getOverlay	Get the address of the overlay RAM segment

Pool System Calls	
kern_createPool	Create a memory pool
kern_setPool	Set a memory pool
kern_allocateBlock	Allocate a memory block
kern_deAllocate	Deallocate a memory block
kern_killPool	Kill a memory pool
kern_getPoolById	Get the memory pool by its Id



4.7.1. µKernel system calls

Target: All

Service system calls

int32_t kern_init(void)

Initialize the **kern** manager.

```
input:      -
output:    status      KKERNNOERR   OK
reentrancy: THREAD-SAFE
```

Call example in C:

```
int32_t      status;
...
status = kern_init();
installMyProcesses();
status = kern_runKernel();
INTERRUPTION_SET;

while (TRUE);
```

```
int32_t kern_getState(void)
```

Get the state of the μKernel.

input:	*state	Ptr on the state
	KNOTREADY	The μKernel is not initialised
	KINITIALIZED	The μKernel is initialised
	KRUNNING	The μKernel is running
output:	status	KKERNNOERR
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint8_t      state;
...
status = kern_getState(&state);
switch (state) {
    case: KNOTREADY    { ... break; }
    case: KINITIALIZED { ... break; }
    case: KINITIALIZED { ... break; }

    ...
int32_t      status;
uint8_t      state;
...
```

int32_t kern_runKernel(void)

Start the multi-process. Before using this call some processes/daemons should be installed.

input:	-		
output:	status	KKERNNOERR	OK
reentrancy:	THREAD-SAFE		

Call example in C:

```
int32_t      status;  
...  
status = kern_init();  
installMyProcesses();  
status = kern_runKernel();  
INTERRUPTION_SET;  
  
while (TRUE);
```

```
int32_t kern_createProcess(const spec_t *specification,
                           const void *argument,
                           volatile proc_t **handle)
```

Create and install a process.

input:	*specification	Ptr on the process specification
	**handle	Ptr on the handle
output:	status	KKERNNOERR KKERNLIFUL KKERNIDPRO
		OK No more process The process Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
#define      KSIZE      512

                  int32_t      status;
static          uint32_t     vStack_0[KSZSTACK];
static volatile proc_t      *vProcess_0, *vProcess_1;
const spec_t    vSpecs_0 = {
    .oIdentifier      = "User Process 0",
    .oText            = "Process to do something (static mode)",
    .oCode             = (const void (*) (const void *argument))process_0,
    .oStkSupBegin     = &vStack_0[0],
    .oStkSup          = &vStack_0[KSZSTACK-8],
    .oSzStk           = KSZSTACK-8,
    .oStkUsr          = NULL,
    .oStackMode        = KSTKSTATIC,
    .oPriority         = 5,
    .oIdPack           = 0
    .oCommManager      = KDEF0,
    .oKind              = KPROCNORMAL;
    .oMode              = KUSER;
};
```

```

uint32_t      *vStack_1= (uint32_t *)((((uint32_t)syos_malloc(KINTERNAL, \
                                         (KSZSTACK* sizeof(uint32_t))+8U))+7U)&(~0x7U));
spec_t         vSpecs_1 = {
    .oIdentifier      = "User Process 1",
    .oText            = "Process to do something (dynamic mode)",
    .oCode             = (const void (*)(const void *argument))process_1,
    .oStkSupBegin     = &vStack_1[0],
    .oStkSup          = &vStack_1[KSZSTACK-8],
    .oSzStk           = KSZSTACK-8,
    .oStkUsr          = NULL,
    .oStackMode        = KSTKDYNAMIC,
    .oPriority         = 5,
    .oIdPack           = 0
    .oCommManager      = KDEFO,
    .oKind              = KPROCNORMAL,
    .oMode              = KUSER;
};

...
status = kern_createProcess(&vSpecs_0, &vProcess_0);
status = kern_createProcess(&vSpecs_1, &vProcess_1);

```

To simplify the way to write this initialization, a couple of macros are available; so, the previous example can be written in this way:

```

#define      KSIZE      512

int32_t      status;
char_t       aStrProcess_0[] = "Process to do something 0\r\n";
char_t       aStrProcess_1[] = "Process to do something 1\r\n";
proc_t       *vProcess_0, *vProcess_1;

...
PROC_USER(0, vSpecs_0, aStrProcess_0, KSIZE, process_0, \
          "User Process 0", KDEFO, 5, 0, KPROCNORMAL)
PROC_USER(1, vSpecs_1, aStrProcess_1, KSIZE, process_1, \
          "User Process 1", KUSBO, 5, 0, KPROCNORMAL)

status = kern_createProcess(&vSpecs_0, NULL, &vProcess_0);
status = kern_createProcess(&vSpecs_1, NULL, &vProcess_1);

```

Notice: the process 0 uses the default communication manager (**KDEF0**) for its I/Os (i.e. the printf). The process 1 uses the usb0 communication manager (**KUSBO**) for its I/Os (i.e. the printf).

int32_t kern_killProcess(volatile proc_t *handle)

Kill a process.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR KKERNREFRS KKERNRESDE KKERNNOPRO
		OK The first process cannot be removed The system daemons cannot be removed The process does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     proc_t      *process;
...
status = kern_killProcess(process);
```

```
int32_t kern_setPriority(volatile proc_t *handle,
                         uint8_t priority)
```

Set a new static priority for a process.

input:	*handle	Ptr on the handle
	priority	Process priority
output:	status	KKERNNOERR OK
		KKERNNOPRO The process does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     proc_t      *process;
...
status = kern_setPriority(process, 23);
```

```
int32_t kern_getProcessByNb(uint8_t number,
                           volatile proc_t **handle)
```

Get the handle of a process by its number.

input:	number	The process number
	**handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOPRO The process does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile proc_t *process;
...
status = kern_getProcessByNb(12, &process);
```

```
int32_t kern_getProcessById(const char_t *identifier,  
                           volatile proc_t **handle)
```

Get the handle of a process by its Id.

input:	* identifier	Ptr on the process Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOPRO The process does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t      identifier[] = "Process 0 - read a sensors";  
volatile     proc_t      *process;  
...  
status = kern_getProcessById(identifier, &process);
```

```
int32_t kern_getProcessRun(volatile proc_t **handle)
```

Get the handle of a process (the running one).

input:	**handle	Ptr on the handle
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile proc_t *process;
...
status = kern_getProcessRun(&process);
```

```
int32_t kern_setGlobalPtr(const void **global)
```

Set a global pointer. This pointer will be updated at every context switching with the contents of the local pointer initialized by a process. The typical usage is the management of the **impure_ptr** necessary for supporting the reentrant implementation of the **newlib**.

```
input:      **global          Ptr global
output:     status           KKERNNOERR  OK
reentrancy: THREAD-SAFE
```

Call example in C:

```
int32_t      status;
void        *global;
...
status = kern_setGlobalPtr(&global);
```

```
int32_t kern_setLocalPtr(const void *local)
```

Set a local pointer. The global pointer initialized by the call **kern_setGlobalPtr** will be updated at every context switching with the contents of the local pointer initialized by a process. The typical usage is the management of the **impure_ptr** necessary to support the reentrant implementation of the **newlib**.

```
input:      *local          Ptr local
output:     status          KKERNNOERR  OK
reentrancy: THREAD-SAFE
```

Call example in C:

```
int32_t      status;
void        *local;
...
status = kern_setLocalPtr(&local);
```

int32_t kern_installCallBack(void (*code)(uint8_t state))

Install a callback routine. The routine is called by the **idle daemon** (when scheduled).

input:	*code	Ptr on the routine code
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
static void _myRoutine(uint8_t state);

int32_t status;
...
status = kern_installCallBack(_myRoutine);
...

// My callback routine

static void _myRoutine(uint8_t state) {

    if (state == KINIDLE) misc_onLed(KLEDIDLE);
    else                  misc_offLed(KLEDIDLE);
}
```

```
int32_t kern_getPC(volatile proc_t **handle,
                    uint32_t *pc)
```

Get the PC of a process (the moment it was de-scheduled).

input:	*handle	Ptr on the handle
	*pc	Ptr on the PC
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     pc;
volatile    proc_t    *process;
...
status = kern_getPC(process, &pc);
```

```
int32_t kern_getFunctionName(const uint32_t pc,
                            const char_t **function)
```

Get the name of a C function. This system call needs the system to be compiled with the option FLAG_CC “-mpoke-function-name” on. For the moment only ARM processors are supported.

input:	pc	Pc of somewhere in the function
	**function	Ptr on the function name
output:	status	KKERNNOERR OK
		KKERNNSYNCNA System call not available
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t      pc;
volatile proc_t *process;
...
status = kern_getPC(process, &pc);
```

Temporal system calls

```
int32_t kern_suspendProcess(uint32_t time)
```

Suspend the process for a time. The time resolution is of **1-ms**. This call allows to suspend a process for a time between ~0-ms to 1193-days.

input:	time	Suspend the process for a time (1-ms of resolution)
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
#define KWAIT312MS 312 // Wait for 312-ms

int32_t status;
...
status = kern_suspendProcess(KWAIT312MS);
```

int32_t kern_waitAtLeast(uint16_t time)

Wait for short times (at least ... a certain time). The time resolution is of 1- μ s. This call allows to suspend a process for a time between ~0- μ to ~65-ms.

input:	time	Suspend the process for a time (1- μ s of resolution)
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
#define KWAITMIN 120 // Wait at least 120- $\mu$ s
int32_t status;
...
status = kern_waitAtLeast(KWAITMIN);
```

int32_t kern_setUnixTime(const atim_t *time)

Set the 32-bit UNIX absolute time/calendar. The time resolution is of **1-s**. This call allows to set the RTC. The time could start the 1st January 1970 at 0h, 0m, 0s (UNIX time = 0x00000000) and can count without a rollover until 2nd July 2106 at 6h, 28m, 16s (UNIX time = 0xFFFFFFFF). The UNIX time is computed and updated in the structure.

input:	*time	Ptr on the time structure
output:	status	KKERNNOERR OK KKERNTIFMT Time format error
reentrancy:	THREAD-SAFE	

Supported structure elements:

```
struct atim {
    uint32_t    oAbsolute;    // Binary absolute time      [0..2^32]
    uint8_t     oSeconds;     // Seconds                  [0..59]
    uint8_t     oMinutes;     // Minutes                 [0..59]
    uint8_t     oHours;       // Hours                   [0..23]
    uint8_t     oWeekDays;    // Week day                [0..6] 0 = Sunday
    uint8_t     oDays;        // Days                    [1..28,29,30,31]
    uint8_t     oMonths;      // Months                  [1..12]
    uint16_t    oYears;       // Years                   [1970..2106]
};
```

Call example in C:

```
atim_t      time;
int32_t     status;
...
time.oYear   = 2017;
time.oMonths = KAUGUST;
time.oDays   = 17;
time.oHours  = 13;
time.oMinutes= 11;
time.oSeconds= 0;
status = kern_setUnixTime(&time);
```

```
int32_t kern_getUnixTime(atim_t *time)
```

Get the 32-bit UNIX absolute time/calendar.

input:	*time	Ptr on the time structure
output:	status	KKERNNOERR
reentrancy:	THREAD-SAFE	OK

Supported structure elements:

```
struct atim {
    uint32_t    oAbsolute;      // Binary absolute time      [0..2^32]
    uint8_t     oSeconds;       // Seconds                  [0..59]
    uint8_t     oMinutes;       // Minutes                 [0..59]
    uint8_t     oHours;         // Hours                   [0..23]
    uint8_t     oWeekDays;      // Week day                [0..6] 0 = Sunday
    uint8_t     oDays;          // Days                    [1..28,29,30,31]
    uint8_t     oMonths;        // Months                  [1..12]
    uint16_t    oYears;         // Years                   [1970..2106]
};
```

Call example in C:

```
atim_t      time;
int32_t     status;
...
status = kern_getUnixTime(&time);
printf("YYYY %d, MM %d, DD %d, WD %d, hh %d, mm %d, ss %d UNIX time %8X\n", \
    time.oYear, time.oMonths, time.oDays, time.oWeekDays, \
    time.oHours, time.oMinutes, time.oSeconds, time.oWeekDays, time.oAbsolute);
```

int32_t kern_switchFast(void)

Force the context switching.

input: -
output: status KKERNNOERR OK
reentrancy: THREAD-SAFE

Call example in C:

```
int32_t status;  
...  
status = kern_switchFast();
```

```
int32_t kern_getTiccount(uint64_t *ticcount)
```

Get the **tic-count** from the start of the μkernel. The **tic-count** is the number of µs intervals from the start of the μKernel. The tic-count uses a 64-bit counter; this allows to represent times between 1-µs to ~584942-years, before having a rollover.

input:	*ticcount	Ptr on the tic-count
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
uint64_t ticcount;
...
status = kern_getTiccount(&ticcount);
```

```
int32_t kern_runPrecise(uint16_t time,
                        void (*code)())
```

Cyclically run a function with a precise temporal constraint. This system call can be useful when temporal precise state machines need to be used. The time resolution is of 1-μs. This call allows to sequence function from ~1-μ to ~65535-μs.

input:	time	Time interval (1-μs of resolution)
	*code	Ptr on the routine code
output:	status	KKERNNOERR
reentrancy:	NO	OK

Call example in C:

```
status = kern_runPrecise(100, (void (*)(void))aStateMachine);

int32_t      status;
...
// My callback routine

static void   aStateMachine(void) {
    static uint32_t      vNb = 0;
    static enum           { KSTATE1, KSTATE2, KSTATE3 } vState = KSTATE1;

    switch (vState) {
        case KSTATE1: {
            if (++vNb < 1)          { break; }
            else                     { vNb = 0; vState = KSTATE2; break; }
        }
        case KSTATE2: {
            if (++vNb < 10)         { break; }
            else                     { vNb = 0; vState = KSTATE3; break; }
        }
        case KSTATE3: {
            if (++vNb < 5)          { break; }
            else                     { vNb = 0; vState = KSTATE1; break; }
        }
    }
}
```

Semaphore system calls

```
int32_t kern_createSyncSemaphore(const char_t *identifier,  
                                volatile sema_t **handle)
```

Create a sync semaphore.

input:	*identifier	Ptr on the semaphore Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNSEFUL No more semaphore
		KKERNIDSEM The semaphore Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "Semaphore 0";  
volatile sema_t *semaphore;  
...  
status = kern_createSyncSemaphore(identifier, &semaphore);
```

```
int32_t kern_createMutxSemaphore(const char_t *identifier,  
                                volatile sema_t **handle)
```

Create a mutx semaphore.

input:	* identifier	Ptr on the semaphore Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNSEFUL No more semaphore
		KKERNIDSEM The semaphore Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t      identifier[] = "Semaphore 0";  
volatile     sema_t      *semaphore;  
...  
status = kern_createMutxSemaphore(identifier, &semaphore);
```

```
int32_t kern_waitSemaphore(volatile sema_t *handle,
                           uint32_t nbEvents)
                           uint32_t timeout)
```

Wait for a semaphore.

input:	*handle	Ptr on the handle
	nbEvents	Number of events
	timeout	Timeout (-1 = infinite time)
output:	status	KKERNNOERR OK
		KKERNNOSEM The semaphore does not exist
		KKERNSENOI The semaphore is not initialized
		KKERNSETME The semaphore counts too many events
		KKERNSEKIL The semaphore has been killed
		KKERNTIMEO Timeout
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     sema_t    *semaphore;
...
status = kern_waitSemaphore(semaphore, 1, timeout);
```

```
int32_t kern_lockSemaphore(volatile sema_t *handle,
                           uint32_t timeout)
```

Lock a semaphore.

input:	*handle	Ptr on the handle
	timeout	Timeout (-1 = infinite time)
output:	status	KKERNNOERR KKERNNOSEM KKERNSENOI KKERNSETME KKERNSEKIL KKERNTIMEO
		OK The semaphore does not exist The semaphore is not initialized The semaphore counts too many events The semaphore has been killed Timeout
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     sema_t      *semaphore;
...
status = kern_lockSemaphore(semaphore, timeout);
// Atomic access code portion
status = kern_unlockSemaphore(semaphore, timeout);
```

```
int32_t kern_signalSemaphore(volatile sema_t *handle)
```

Signal a semaphore.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR KKERNNOSEM KKERNSENOI KKERNSETME
		OK The semaphore does not exist The semaphore is not initialized The semaphore counts too many events
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     sema_t      *semaphore;
...
status = kern_signalSemaphore(semaphore);
```

```
int32_t kern_unlockSemaphore(volatile sema_t *handle)
```

Unlock a semaphore.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR KKERNNOSEM KKERNSENOI KKERNSETME
		OK The semaphore does not exist The semaphore is not initialized The semaphore counts too many events
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     sema_t      *semaphore;
...
status = kern_lockSemaphore(semaphore, timeout);
// Atomic access code portion
status = kern_unlockSemaphore(semaphore, timeout);
```

```
int32_t kern_killSemaphore(volatile sema_t *handle)
```

Kill a semaphore.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOSEM The semaphore does not exist KKERNSENOI The semaphore is not initialized
reentrancy: THREAD-SAFE		

Call example in C:

```
int32_t      status;
volatile     sema_t      *semaphore;
...
status = kern_killSemaphore(semaphore);
```

```
int32_t kern_restartSemaphore(volatile sema_t *handle)
```

Restart a semaphore.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR KKERNSENOI KKERNNOSEM KKERNSEPRP
		OK The semaphore is not initialized The semaphore does not exist A process is still connected to the list
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     sema_t      *semaphore;
...
status = kern_restartSemaphore(semaphore);
```

```
int32_t kern_getSemaphoreById(const char_t *identifier,  
                           volatile sema_t **handle)
```

Get the handle of a semaphore by its Id.

input:	* identifier	Ptr on the semaphore Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOSEM The semaphore does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t    identifier[] = "Semaphore 0";  
volatile     sema_t    *semaphore;  
...  
status = kern_getSemaphoreById(identifier, &semaphore);
```

Mailbox/queue system calls

```
int32_t kern_createMailbox(const char_t *identifier,  
                           volatile mbox_t **handle)
```

Create a mailbox.

input:	*identifier	Ptr on the mailbox Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR KKERNMBFUL KKERNIDMBO
		OK No more mailbox The mailbox Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "Mailbox 0";  
volatile mbox_t *mailBox;  
...  
status = kern_createMailbox(identifier, &mailBox);
```

```
int32_t kern_writeMailbox(volatile mbox_t *handle,
                           uint8_t *message,
                           uint16_t size)
```

Write a message in the mailbox.

input:	*handle	Ptr on the handle
	*message	Ptr on the message
	size	Size of the message
output:	status	KKERNNOERR OK
		KKERNNOMBO The mailbox does not exist
		KKERNMBNOI The mailbox is not initialized
		KKERNTOPAC The mailbox is full
reentrancy:	THREAD-SAFE	

Call example in C:

```
#define      KSIZE      5

                  int32_t      status;
uint8_t      message[KSIZE] = {1,2,3,4,5};
volatile     mbox_t      *mailBox;
...
status = kern_writeMessage(mailBox, message, KSIZE);
```

```
int32_t kern_writeQueue(volatile mbox_t *handle,
                        uint32_t message)
```

Write a message in the queue.

input:	*handle	Ptr on the handle
	message	Message
output:	status	KKERNNOERR KKERNNOMBO KKERNMBNOI KKERNTOPAC
		OK The mailbox does not exist The mailbox is not initialized The mailbox is full
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint8_t       message = 0x2345;
volatile mbox_t *mailBox;
...
status = kern_writeQueue(mailBox, message);
```

```
int32_t kern_readMailbox(volatile mbox_t *handle,
                         uint8_t **message,
                         uint16_t *size)
```

Read a message from the mailbox.

input:	* handle	Ptr on the handle
	** message	Ptr on the message
	* size	Ptr on the Size of the message
output:	status	OK
	KKERNNOERR	The mailbox does not exist
	KKERNNOMBO	The mailbox is not initialized
	KKERNMBNOI	The mailbox is empty
	KKERNNOPAC	The mailbox is empty
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint8_t      size, *message;
volatile     mbox_t      *mailBox;
...
status = kern_readMessage(mailBox, &message, &size);
```

```
int32_t kern_readQueue(volatile mbox_t *handle,
                      uint32_t *message)
```

Read a message from the queue.

input:	*handle	Ptr on the handle
	*message	Ptr on the message
output:	status	KKERNNOERR KKERNNOMBO KKERNMBNOI KKERNNPAC
		OK The mailbox does not exist The mailbox is not initialized The mailbox is empty
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     message;
volatile     mbox_t    *mailBox;
...
status = kern_readQueue(mailBox, &message);
```

```
int32_t kern_killMailbox(volatile mbox_t *handle)
```

Kill a mailbox.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR KKERNNOSEM KKERNSENOI KKERNPAPRP
		OK The mailbox does not exist The mailbox is not initialized A message is still inside the mailbox
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     mbox_t      *mailBox;
...
status = kern_killMailbox(mailBox);
```

```
int32_t kern_getMailboxById(const char_t *identifier,  
                           volatile mbox_t **handle)
```

Get the handle of a mailbox by its Id.

input:	* identifier	Ptr on the mailbox Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOMBO The mailbox does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t      identifier[] = "Mailbox 0";  
volatile     mbox_t      *mailBox;  
...  
status = kern_getMailboxById(identifier, &mailBox);
```

Association system calls

```
int32_t kern_createAssociation(const char_t *identifier,  
                                volatile astp_t **handle)
```

Create an association.

input:	*identifier	Ptr on the association Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNASFUL No more association
		KKERNIDASS The association Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "Association 0";  
volatile astp_t *association;  
...  
status = kern_createAssociation(identifier, &association);
```

```
int32_t kern_publishAssociation(volatile astp_t *handle,
                                shar_t *pack)
```

Publish an association.

input:	*handle	Ptr on the handle
	*pack	Ptr on the data pack association
output:	status	KKERNNOERR OK
		KKERNNOASS The association does not exist
		KKERNASNOI The association is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     sharedBuffer[123];
shar_t       pack;
volatile astp_t *association;
...
pack.oGeneral = (void *)&sharedBuffer[0];
pack.oSize = sizeof(sharedBuffer);
status = kern_publishAssociation(association, &pack);
```

```
int32_t kern_findAssociation(volatile astp_t *handle,
                             shar_t **pack)
```

Find an association.

input:	*handle	Ptr on the handle
	**pack	Ptr on the data pack association
output:	status	KKERNNOERR OK
		KKERNNOASS The association does not exist
		KKERNASNOI The association is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     *counter, size;
shar_t       *pack;
volatile astp_t *association;
...
status = kern_findAssociation(association, &pack);
counter = (uint32_t *)pack->oGeneral;
size = pack->oSize;
```

```
int32_t kern_killAssociation(volatile astp_t *handle)
```

Kill an association.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOASS The association does not exist KKERNASNOI The association is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile astp_t *association;
...
status = kern_killAssociation(association);
```

```
int32_t kern_getAssociationById(const char_t *identifier,  
                                volatile astp_t **handle)
```

Get the handle of an association by its Id.

input:	* identifier	Ptr on the association Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOASS The association does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t      identifier[] = "Association 0";  
volatile     astp_t      *association;  
...  
status = kern_getAssociationById(identifier, &association);
```

Software event system calls

```
int32_t kern_createEvent(const char_t *identifier,  
                         volatile evnt_t **handle)
```

Create an event.

input:	*identifier	Ptr on the event Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR KKERNEVFUL KKERNIDEVO
		OK No more event The event Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "BTLE message";  
volatile evnt_t *event;  
...  
status = kern_createEvent(identifier, &event);
```

```
int32_t kern_getBitEvent(volatile evnt_t *handle,
                         uint32_t *bitEvent)
```

Get the bit event.

input:	*handle	Ptr on the handle
	*bitEvent	Ptr on the bit event
output:	status	KKERNNOERR OK
		KKERNNOEVO The event does not exist
		KKERNEVNOI The event is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     bitEvent;
volatile    evnt_t   *event;
...
status = kern_getBitEvent(event, &bitEvent);
```

```
int32_t kern_signalEvent(uint32_t bitEvent,
                         uint32_t mode)
```

Signal one events. This is used to have a software synchronization between processes.

input:	bitEvent	Event (or events)
	mode	KEVNTSWTC Synchro with a context switching
		KEVNTCONT Synchro without a context switching
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     bitEvent;
volatile evnt_t *event;
...
kern_getBitEvent(event, &bitEvent);
status = kern_signalEvent(bitEvent, KSYNCCONT);
```

```
int32_t kern_waitEvent(uint32_t *bitEvent,
                      uint32_t timeout)
```

Waiting for one or more events. This is used to have a software synchronization between processes.

```
input:      *bitEvent          Ptr on the events
            timeout           Timeout (-1 = infinite time)
output:     status             KKERNNOERR   OK
                  KKERNNEVKIL The event has been killed
                  KKERNTIMEO  Timeout
reentrancy: THREAD-SAFE
```

Call example in C:

```
int32_t      status;
uint32_t      eventTMP, eventMSG, eventXFER;
volatile evnt_t *event;
...
status = kern_getEventById("message OK", &event);
status = kern_getBitEvent(event, &eventMSG);
status = kern_getEventById("start", &event);
status = kern_getBitEvent(event, &eventXFER);
eventTMP = eventMSG | eventXFER;
status = kern_waitEvent(&eventTMP, KSYNCCONT);
```

```
int32_t kern_killEvent(volatile evnt_t *handle)
```

Kill an event.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOEVO The event does not exist KKERNEVNOI The event is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile evnt_t *event;
...
status = kern_killEvent(event);
```

```
int32_t kern_getEventById(const char_t *identifier,  
                           volatile evnt_t **handle)
```

Get the handle of an event by its Id.

input:	* identifier	Ptr on the event Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOEVO The event does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t      identifier[] = "Event 0";  
volatile     evnt_t      *event;  
...  
status = kern_getEventById(identifier, &event);
```

Software signal system calls

```
int32_t kern_createSignal(const char_t *identifier,  
                           volatile sign_t **handle)
```

Create a signal.

input:	*identifier	Ptr on the signal Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNISFUL No more signal
		KKERNIDSIO The signal Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "BTLE message";  
volatile sign_t *signal;  
...  
status = kern_createSignal(identifier, &signal);
```

```
int32_t kern_getBitSignal(volatile sign_t *handle,
                           uint32_t *bitSignal)
```

Get the bit signal.

input:	*handle	Ptr on the handle
	*bitSignal	Ptr on the bit signal
output:	status	KKERNNOERR OK
		KKERNNOSIO The signal does not exist
		KKERNSINOI The signal is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
uint32_t     bitSignal;
volatile sign_t *signal;
...
status = kern_getBitSignal(signal, &bitSignal);
```

```
int32_t kern_signalSignal(volatile sign_t *handle,
                           uint32_t bitSignal,
                           volatile proc_t *process,
                           uint32_t mode)
```

Signal one or more signal. This is used to have a software synchronization between processes. The mechanism can be selective (specific to one process), or it could be a broadcast (to all the installed processes).

input:	*handle	Ptr on the handle
	bitSignal	Signal (or signals)
	*process	Ptr on the process handle for the selective
mode		
	NULL	For the broadcast mode
	mode	KSIGNSWTC Synchro with a context switching
		KSIGNCONT Synchro without a context switching
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```

int32_t      status;
uint32_t     bitSignal;
volatile    sign_t      *signal;
volatile    proc_t      *process;
...
while (kern_getProcessById("My process", &process) != KKERNNOERR) {
    kern_suspendProcess(1);
}
...
// Mode selective

status = kern_signalSignal(signal, bitSignal, process, KSYNCCONT);
...
// Mode broadcast
...
status = kern_signalSignal(signal, bitSignal, NULL, KSYNCCONT);
```

```
int32_t kern_waitSignal(uint32_t *bitSignal,
                        uint32_t timeout)
```

Waiting for one or more signals. This is used to have a software synchronization between processes.

input:	*bitSignal	Ptr on the signal
	timeout	Timeout (-1 = infinite time)
output:	status	KKERNNOERR OK
		KKERNSIKIL The signal has been killed
		KKERNTIMEO Timeout
reentrancy:	THREAD-SAFE	

Call example in C:

```
#define KSIGNAL_MSG 0x00080000

        int32_t status;
volatile sign_t *signal;
...
status = kern_getSignalById("message OK", &signal);
status = kern_waitSignal(KSIGNAL_MSG, KSIGNCONT);
```

```
int32_t kern_killSignal(volatile sign_t *handle)
```

Kill a signal.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOSIO The signal does not exist KKERNNSINOI The signal is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile sign_t *signal;
...
status = kern_killSignal(signal);
```

```
int32_t kern_getSignalById(const char_t *identifier,  
                           volatile sign_t **handle)
```

Get the handle of a signal by its Id.

input:	* identifier	Ptr on the signal Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOSIO The signal does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t      identifier[] = "Signal 0";  
volatile     sign_t      *signal;  
...  
status = kern_getSignalById(identifier, &signal);
```

Software timer system calls

```
int32_t kern_createSoftwareTimer(const char_t *identifier,  
                                volatile stim_t **handle)
```

Create a software timer.

input:	*identifier	Ptr on the event Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR KKERNSTFUL KKERNIDSTI
		OK No more software timer The software timer Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "Timer acquisition";  
volatile stim_t *softwareTimer;  
...  
status = kern_createSoftwareTimer(identifier, &softwareTimer);
```

```
int32_t kern_setSoftwareTimer(volatile stim_t *handle,
                             tspc_t *configure)
```

Set (configure) a software timer.

input:	*handle	Ptr on the handle
	*configure	Ptr on the configure
output:	status	KKERNNOERR OK
		KKERNNOSTI The software timer does not exist
		KKERNSTNOI The software timer is not initialized
reentrancy:	THREAD-SAFE	

Supported structure elements:

```
struct tspc {
    uint8_t      oMode;                      // Mode
    #define       KSINGLESHOT   0               // Single shot software timer
    #define       KCONTINUE     1               // Continue software timer

    uint32_t     oInitTime;                  // Initial time
    uint32_t     oTime;                     // Continuous time
    const void   *oArgument;                // Ptr on the timer argument
    void         (*oCode)(void);            // Ptr on the code
};

struct stim {
    const char_t *oIdentifier;              // Software timer identifier
    uint8_t      oState;                   // Status
    #define       BSTIMINSTALLED  0           // Software timer installed
    #define       BSTIMEEXECUTED 1           // Software timer executed
    #define       BSTIMRUNNING   2           // Software timer is running
    #define       BSTIMEEXECUTED 3           // Software timer executed

    uint32_t     oInitCounter;             // Initial time (decremented)
    uint32_t     oCounter;                 // Continuous time (decremented)
    tspc_t       oTimerSpec;              // Software timer specifications
};
```

Call example in C:

```
static void _changeStateLed(void);

    int32_t      status;
    tspc_t       configure;
volatile stim_t   *softwareTimer;
...
configure.oMode     = KCONTINUE;
configure.oInitTime = 10000;
configure.oTime     = 1000;
configure.oArgument = NULL;
configure.oCode      = _changeStateLed;
...
status = kern_setSoftwareTimer(softwareTimer, &configure);
...

// My callback routine

static void _changeStateLed(void) {

    misc_toggleLed(0);
}
```

```
int32_t kern_killSoftwareTimer(volatile stim_t *handle)
```

Kill a software timer.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOSTI The software timer does not exist KKERNSTNOI The software timer is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile stim_t *softwareTimer;
...
status = kern_killSoftwareTimer(softwareTimer);
```

```
int32_t kern_getSoftwareTimerById(const char_t *identifier,  
                                  volatile stim_t **handle)
```

Get the handle of a software timer by its Id.

input:	* identifier	Ptr on the event Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOSTI The software timer does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t    identifier[] = "Timer acquisition";  
volatile     stim_t   *softwareTimer;  
...  
status = kern_getSoftwareTimerById(identifier, & softwareTimer);
```

Memory pool system calls

```
int32_t kern_createPool(const char_t *identifier,  
                      volatile pool_t **handle)
```

Create a memory pool.

input:	*identifier	Ptr on the signal Id
	**handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNPOFUL No more memory pool
		KKERNIDPOI The memory pool Id is already used
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;  
const char_t identifier[] = "Xfer buffer";  
volatile pool_t *memoryPool;  
...  
status = kern_createPool(identifier, &memoryPool);
```

```
int32_t kern_setPool(volatile pool_t *handle,
                      pcnf_t *configure)
```

Configure a memory pool.

input:	*handle	Ptr on the handle
	*configure	Ptr on the configure
output:	status	KKERNNOERR KKERNNOPOI KKERNPONOI KKERNSTNOI
		OK The memory pool does not exist The memory pool is not initialized The memory pool configuration is not possible
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
pcnf_t       configure;
volatile pool_t *memoryPool;
...
configure.oNbBlocks = 16;
configure.oBlockSize = 256;
configure.oLocation = KINTERNAL;
status = kern_setPool(memoryPool, &configure);
```

```
int32_t kern_allocateBlock(volatile pool_t *handle,
                           void **address)
```

Allocate a block of the memory pool.

input:	*handle	Ptr on the handle
	**address	Ptr on the block address
output:	status	KKERNNOERR OK
		KKERNNOPOI The memory pool does not exist
		KKERNPONOI The memory pool is not initialized
		KKERNBKFUL No more block
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
void         *address;
volatile     pool_t   *memoryPool;
...
status = kern_allocateBlock(memoryPool, &address);
```

```
int32_t kern_deAllocateBlock(volatile pool_t *handle,
                             void *address)
```

Allocate a block of the memory pool.

input:	*handle	Ptr on the handle
	*address	Ptr on the block address
output:	status	KKERNNOERR KKERNNOPOI KKERNPONOI KKERNNOBKI
		OK The memory pool does not exist The memory pool is not initialized The block does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
void         *address;
volatile     pool_t   *memoryPool;
...
status = kern_deAllocateBlock(memoryPool, address);
```

int32_t kern_killPool(volatile pool_t *handle)

Kill a memory pool.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOPOI The memory pool does not exist KKERNPONOI The memory pool is not initialized
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;
volatile     pool_t      *memoryPool;
...
status = kern_killPool(memoryPool);
```

```
int32_t kern_getPoolById(const char_t *identifier,  
                         volatile sign_t **handle)
```

Get the handle of a memory pool by its Id.

input:	* identifier	Ptr on the memory pool Id
	** handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNNOPOI The memory pool does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
const        char_t    identifier[] = "Xfer buffer";  
volatile     pool_t    *memoryPool;  
...  
status = kern_getPoolById(identifier, &memoryPool);
```

Overlay system calls**int32_t kern_loadOverlay(uint32_t idModule)**

Load an overlay module inside the overlay RAM segment.

input:	idModule	"XXYY" module Id
output:	status	KKERNNOERR OK
		KKERNOVLRY The overlay process does not exist
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t      status;  
...  
status = kern_loadOverlay("OV01");
```

```
int32_t kern_getOverlay(void **address)
```

Get the address of the overlay RAM segment.

input:	**address	Ptr on the overlay process entry address
output:	status	KKERNNOERR OK
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
void *address;
...
status = kern_getOverlay(&address);
```

Debug system calls

int32_t kern_stopProcess(volatile proc_t *handle)

Stop a process and place it in the debug list.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK
		KKERNDBGERR The process is already in the debug list
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile proc_t *process
...
status = kern_stopProcess(process);
```

```
int32_t kern_reactivateProcess(volatile proc_t *handle)
```

Reactivate a process located in the debug list.

input:	*handle	Ptr on the handle
output:	status	KKERNNOERR OK KKERNNOPRO The process does not exist KKERNDBGER The process is not in the debug list
reentrancy:	THREAD-SAFE	

Call example in C:

```
int32_t status;
volatile proc_t *process
...
status = kern_reactivateProcess(process);
```

4.8. References

David Décotigny, “Une infrastructure de simulation modulaire pour l’ évaluation de performances de systèmes temps-réel”, Thèse de doctorat de l’Université de Rennes 1, 2003.

Larisa Rizvanovic, “Comparison between Real Time Operating Systems in Hardware and Software”, Thesis presented to Mälardalens University, 2001.

J. Stankovic, M. Spuri, M. DiNatale, and G. Buttazzo, “Implications of classical scheduling results for real-time systems”, Technical Report UM- CS-1994-089, U. Mass, 1994.

A. Colin and I. Puaut, “Analyse de temps d’exécution au pire cas du système d’exploitation temps-réel RTEMS”, In Seconde Conférence Française sur les Systèmes d’Exploitation (CFSE2), pages 73-84, April 2001.

G. Buttazzo and J. Stankovic, “Red: Robust earliest deadline scheduling”, In Proceedings of the International Workshop on Responsive Computing Systems, 1993.

Reeves, Glenn, "What Really Happened on Mars?", Risks-Forum Digest, Volume 19: Issue 58, January 1998.

Sha L., R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Transactions on Computers, September 1990, p. 1175.

Kalinsky, David and Michael Barr, "Priority Inversion" Embedded Systems Programming, April 2002, pp. 55-56.

Jerry Farmer, "A Real-Time Operating System for PICmicroTM Microcontrollers" Microchip AN585 1997.

Keil Software, "RTX51 Tiny, 8051 Real-Time OS" Keil Software, 1995.

4.9. Links

<http://www.gnu.org/>

<http://lists.gnu.org/>

<http://sourceware.org/newlib/>

<http://www.embedded.com/>

<http://www.netrino.com/Publications/>

<http://www.billgatliff.com/>

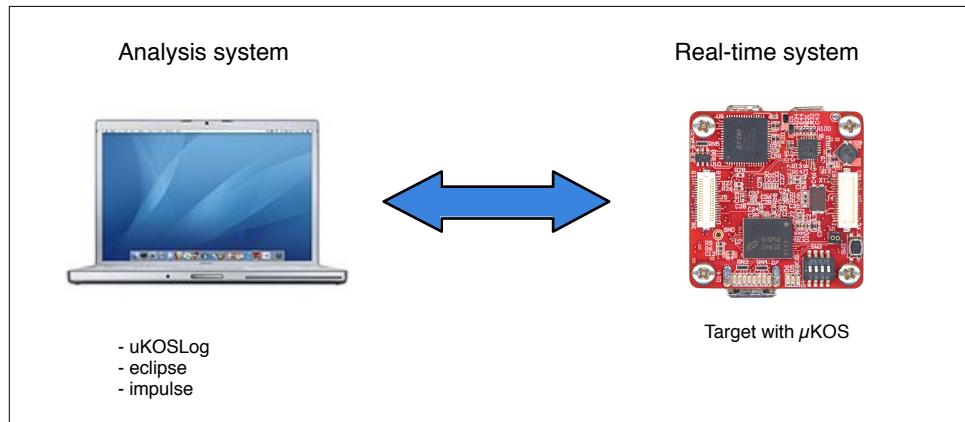
v 2.9

5. Event analysis



5.1. Event analysis

When operating with multitasking μKernel, it is sometimes necessary to display events than can occur rarely (once per day or once after millions of system events, etc.). Catching these “**single events**” can be tricky (we can for example use a logical analyzer and synchronize it on the change of a GPIO to trap the event). A better solution is to use data loggers capable to record all the system events. **Impulse** is a beautiful **Eclipse plugin** designed by the **Toem** team. **Impulse** is capable to read, analyze μKOS events and display them with plenty of associated information on the screen.



Before recording a sequence of events, it is necessary to have in place the following tools and to include in the program a dedicated macro for tagging the wanted events:

- **Eclipse:** this tool (environment) can be downloaded for free: <https://www.eclipse.org>.
- **Impulse:** this eclipse plug-in can be downloaded for free: <http://www.toem.de>.
- **uKOSLog:** this tool is part of the μKOS project.

5.1.1. The tool uKOSLog

This UNIX tool is responsible for recording in a file all the events generated by the target. Before running the program in the target, it is mandatory to run this tool.

```
uKOSlog [driver] [driverID] [baudrate] [nbEvents] [log_file]
driver          VCP or D2XX FTDI driver    i.e. -vcp or -d2x
driverID        driverID                  i.e. DBXIMUEF
baudrate       serial baudrate          i.e. 460800
log_file        the xyz.log file        i.e. /Users/efr/Desktop/mylog.imp
```

```
/*
; uKOSlog.
; =====

;-----
; Author:      Franzi Edo.  The 2014-01-12
; Modifs:
;
; SVN:
; $Author:: efr           $: Author of last commit
; $Rev:: 109             $: Revision of last commit
; $Date:: 2014-08-24 19:27:47#$: Date of last commit
;
; Project:     uKOS
; Goal:        Generate a log file. This is useful to supervise the uKernl
;               activity.
;
; Usage:       uKOSlog [driver] [driverID] [baudrate] [nbEvents] [log_file]
;               driver      input -vcp or -d2x
;               driver id   input serial input device "FT000001"
;               baudrate   input serial baudrate
;               log_file    output the xyz.log file
;
; (c) 1992-2017, Franzi Edo.
; -----
;

; Franzi Edo.          _ _/_ /_ \ \_/
; 5-Route de Cheseaux  / / / / ,< / / / / \_ \
; CH 1400 Cheseaux-Noréaz / / / / | / / / / \_ /
;                         \_,/_/ |_\_//__/
; edo.franzi@ukos.ch

;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <unistd.h>
#include      <fcntl.h>
#include      <sys/stat.h>
#include      <types_unix.h>
#include      <ftd2xx.h>

enum { KVCP, KD2X };
enum { KERRNOT, KERRAPP, KERRCLI };

// Prototypes
// =====

extern void          vcp_list(void);
extern int32_t        vcp_open(const char_t *driverID, int32_t baudrate);
extern void          vcp_close(int32_t handle);
extern void          vcp_readAByte(int32_t handle, uint8_t *receive);
extern void          vcp_read(int32_t handle, uint8_t *receive, uint32_t *count);
extern void          vcp_flush(int32_t handle);

extern void          d2x_list(void);
extern FT_HANDLE     d2x_open(const char_t *driverID, int32_t baudrate);
extern void          d2x_close(FT_HANDLE handle);
extern void          d2x_readAByte(FT_HANDLE handle, uint8_t *receive);
extern void          d2x_read(FT_HANDLE handle, uint8_t *receive, uint32_t *count);
extern void          d2x_flush(FT_HANDLE handle);

static int32_t       _app_vcp(char_t *driverID, char_t *fileName, uint32_t baudrate);
static int32_t       _app_d2x(char_t *driverID, char_t *fileName, uint32_t baudrate);
static void          _log_vcp(int32_t handle, FILE *fpout);
static void          _log_d2x(FT_HANDLE handle, FILE *fpout);

int    main(int argc, char_t *argv[]) {
    char_t          *fileName, *driver;
    char_t          driverNumber[50];
    int32_t         error;
    uint32_t        mode, baudrate;

    fprintf(stdout, __DATE__ " " __TIME__ " (c) EFr-2017\n");
```

```
// Analyze the command line
// -----
//
// Example:
//
// uKOSlog -vcp DCWZ1FL8 460800 output.log
// uKOSlog -d2x DCWZ1FL8 460800 output.log

    error = KERRNOT;
    switch (argc) {
        case 5: {
            if      (strcmp(argv[1], "-vcp") == 0) mode = KVCP; // vcl
            else if (strcmp(argv[1], "-d2x") == 0) mode = KD2X; // d2x
            else error = KERRCLI;

            strcpy(driverID, argv[2]);                                // Driver
            baudrate = (uint32_t)strtol(argv[3], KNULL, 10);          // Baudrate
            fileName = argv[4];                                       // Filename
            break;
        }

        default: error = KERRCLI; break;
    }
    if (error == KERRNOT) {
        switch (mode) {
            case KVCP: error = _app_vcp(driverID, fileName, baudrate); break;
            case KD2X: error = _app_d2x(driverID, fileName, baudrate); break;
        }
    }

    switch (error) {
        case KERRNOT: {
            fprintf(stdout, "Terminated\n");
            return (EXIT_SUCCESS);
        }
        case KERRAPP: {
            fprintf(stderr, "## Error: input/output file not found\n");
            return (EXIT_FAILURE);
        }
        case KERRCLI: {
            fprintf(stderr, "%s %s %s %s %s\n", argv[0], ... argv[4]);
            fprintf(stderr, "uKOSlog [... ... file_name]\n");
            fprintf(stderr, "[driver]           -vcp\n");
            fprintf(stderr, "[driver]           -d2x\n");
            fprintf(stderr, "[driverID]         driver ID (ie. FT000001)\n");
            fprintf(stderr, "[baudrate]         serial baudrate\n");
            fprintf(stderr, "[file_name]        the file_name file\n");
            d2x_list();
            return (EXIT_FAILURE);
        }
    }
}
```

```
        }
    }
// Local routines
// =====

/*
 * \brief _app_vcp
 *
 * - Open the driver & the file
 * - Fill the file
 * - Close the driver & the file
 *
 */
static int32_t      _app_vcp(char_t *driverID, char_t *fileName, uint32_t baudrate) {
    int32_t      handle;
    FILE         *fpout;

    handle = vcp_open(driverID, baudrate); fpout = fopen(fileName, "w");
    if ((handle == KNULL) || (fpout == KNULL)) {
        vcp_close(handle); fclose(fpout);
        return (KERRAPP);
    }

// Fill the file

    vcp_flush(handle);
    _log_vcp(handle, fpout);
    vcp_close(handle); fclose(fpout);
    return (KERRNOT);
}
```

```
/*
 * \brief _app_d2x
 *
 * - Open the driver & the file
 * - Fill the file
 * - Close the driver & the file
 *
 */
static int32_t      _app_d2x(char_t *driverID, char_t *fileName, uint32_t baudrate) {
    FT_HANDLE     handle;
    FILE          *fpout;

    handle = d2x_open(driverID, baudrate); fpout = fopen(fileName, "w");
    if ((handle == KNULL) || (fpout == KNULL)) {
        d2x_close(handle); fclose(fpout);
        return (KERRAPP);
    }

    // Fill the file

    d2x_flush(handle);
    _log_d2x(handle, fpout);
    d2x_close(handle); fclose(fpout);
    return (KERRNOT);
}

/*
 * \brief _log_vcp
 *
 * - Generate the log file
 *
 */
static void  _log_vcp(int32_t handle, FILE *fpout) {
    uint8_t       byte;

    while (TRUE) {

//      32      Header

        if (vcp_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (vcp_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (vcp_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (vcp_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
    }
}
```

```
//      8      Process ID

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      8      Process priority

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      16     Reserve (Alignment)

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      32     Process Stack

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      32     Process State

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
```

```
//      64      Temporal tag us

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      64      Nb of executions of the process

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      64      Cumulated time used by the process

    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
```

```
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (vcp_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
}
}

/*
 * \brief _log_d2x
 *
 * - Generate the log file
 *
 */
static void _log_d2x(FT_HANDLE handle, FILE *fpout, uint32_t nbEvents) {
    uint8_t byte;

    while (TRUE) {

//      32      Header

        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);

//      8      Process ID

        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);

//      8      Process priority

        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);

//      16     Reserve (Alignment)

        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
        if (d2x_readAByte(handle, &byte)) break;
        fwrite((void *)&byte, 1, 1, fpout);
```

```
//      32      Process Stack

    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      32      Process State

    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      64      Temporal tag us

    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
```

```
//      64      Nb of executions of the process

    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

//      64      Cumulated time used by the process

    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);
    if (d2x_readAByte(handle, &byte)) break;
    fwrite((void *)&byte, 1, 1, fpout);

}

}
```

```
/*
; vcp.
; =====

;-----
; Author:      Franzi Edo.  The 2014-01-12
; Modifs:
;
; SVN:
; $Author:: efr           $: Author of last commit
; $Rev:: 109             $: Revision of last commit
; $Date:: 2014-08-24 19:27:47#$: Date of last commit
;
; Project:     uKOS
; Goal:        VCP I/O management.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          _____/_____\_____
;   5-Route de Cheseaux    / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / / \_ / /
;                           \_,/_\_|_\____//____/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <fcntl.h>
#include    <termios.h>
#include    <unistd.h>
#include    <types_unix.h>
#include    <ftd2xx.h>
```

```
typedef     struct termios      termios_t;

static termios_t    vTTYAttr;

/*
 * \brief vcp_list
 *
 * - List of the devices
 *
 */

void  vcp_list(void) {
    FT_STATUS      status;
    DWORD          nbDevices;
    uint32_t       device;
    char           buffer[64];

    status = FT_ListDevices(&nbDevices, NULL, FT_LIST_NUMBER_ONLY);
    if (status == FT_OK) fprintf(stdout, "Number of device: %0d\n", nbDevices);
    else                  fprintf(stdout, "status = %0d\n", status);

    for (device = 0; device < nbDevices; device++) {
        status = FT_ListDevices((PVOID)((intptr_t)device), buffer, \
                               FT_LIST_BY_INDEX | FT_OPEN_BY_SERIAL_NUMBER);
        if (status == FT_OK) fprintf(stdout, "Driver ID: %s\n", buffer);
        else                  fprintf(stdout, "status = %0d\n", status);
    }
}
```

```
/*
 * \brief vcp_open
 *
 * - Open the device
 *
 * \param[in]      *driverID      Ptr on the driver ID "FTXG86D7D"
 * \param[in]      baudrate      Device baudrate
 * \param[out]     handle        Driver handle ID
 *
 */
int32_t      vcp_open(const char_t *driverID, int32_t baudrate) {
    int32_t      handle;
    speed_t      bd;
    termios_t    options;
    char_t       driver[50] = "/dev/tty.usbserial-";

    strcat(driver, driverID);
    switch (baudrate) {
        case 9600:   bd = B9600;   break;
        case 19200:  bd = B19200;  break;
        case 38400:  bd = B38400;  break;
        case 115200: bd = B115200; break;
        case 230400: bd = B230400; break;
        case 460800: bd = B460800; break;
        default:     bd = B460800; break;
    }

    handle = open(driver, O_RDONLY | O_NOCTTY | O_NDELAY);
    if (handle == -1) return (EXIT_FAILURE);

    while (TRUE) {
        if (fcntl(handle, F_SETFL, 0)          == -1) break;
        if (tcgetattr(handle, &vTTYAttr)       == -1) break;

        options      = vTTYAttr;
        options.c_cflag |= (CLOCAL | CREAD);
        options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
        options.c_oflag &= ~OPOST;
        options.c_cc[VMIN] = 0;
        options.c_cc[VTIME] = 5;
        options.c_cflag &= ~(CSIZE | PARENB);
        options.c_cflag |= (CS8 | CSTOPB);

        if (cfsetispeed(&options, bd)         == -1) break;
        if (cfsetspeed(&options, bd)          == -1) break;
        if (tcsetattr(handle, TCSANOW, &options) == -1) break;
        return (handle);
    }
    close(handle);
    return (EXIT_FAILURE);
}
```

```
}

/*
 * \brief vcp_close
 *
 * - Close the device
 *
 * \param[in]      handle      Driver handle ID
 * \param[out]     -
 *
 */
void    vcp_close(int32_t handle) {

    tcsetattr(handle, TCSADRAIN, &vTTYAttr);
    close(handle);
}

/*
 * \brief vcp_readAByte
 *
 * - Read a byte on the device
 *
 * \param[in]      handle      Driver handle ID
 * \param[in]      *receive    Ptr on the receive buffer
 * \param[out]     status      FALSE -> OK
 *                           TRUE -> timeout error
 *
 */
bool_t vcp_readAByte(int32_t handle, uint8_t *receive) {
    int32_t      nbRead;
    uint8_t      byte;
    uint32_t      timeout = 0;

    while (TRUE) {
        nbRead = read(handle, &byte, 1);
        if (nbRead > 0) break;
        usleep(10);
        if (timeout++ > 500000) return (TRUE);
    }

    *receive = byte;
    return (FALSE);
}
```

```
/*
 * \brief vcp_read
 *
 * - Read a byte on the device
 *
 * \param[in]      handle      Driver handle ID
 * \param[in]      *receive    Ptr on the receive buffer
 * \param[out]     status      FALSE -> OK
 *                           TRUE -> timeout error
 *
 */
bool_t vcp_read(int32_t handle, uint8_t *receive, uint32_t *count) {
    int32_t        nbRead;
    uint8_t        byte;
    uint32_t       timeout = 0;

    while (TRUE) {
        nbRead = read(handle, &byte, 100);
        if (nbRead > 0) break;
        usleep(10);
        if (timeout++ > 500000) return (TRUE);
    }

    *count = nbRead;
    return (FALSE);
}

/*
 * \brief vcp_flush
 *
 * - Flush the device
 *
 * \param[in]      handle      Driver handle ID
 * \param[out]     -
 *
 */
void vcp_flush(int32_t handle) {
    int32_t        nbRead;
    uint8_t        byte;

    do {
        nbRead = read(handle, &byte, 1);
    } while (nbRead == 1);
}
```

```
/*
; d2XX.
; =====

;-----
; Author:      Franzi Edo.  The 2014-01-12
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 109             $: Revision of last commit
; $Date::: 2014-08-24 19:27:47#$: Date of last commit
;
; Project:     uKOS
; Goal:        D2XX I/O management.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          _____/_____\_____
;   5-Route de Cheseaux    / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \_\_ / /
;                           \_,/_\_|_\_\_//\_\_/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <unistd.h>
#include      <types_unix.h>
#include      <ftd2xx.h>
```

```
/*
 * \brief d2x_list
 *
 * - List of the devices
 *
 */
void d2x_list(void) {
    FT_STATUS     status;
    DWORD         nbDevices;
    uint32_t      device;
    char          buffer[64];

    status = FT_ListDevices(&nbDevices, NULL, FT_LIST_NUMBER_ONLY);
    if (status == FT_OK) fprintf(stdout, "Number of device: %d\n", nbDevices);
    else                  fprintf(stdout, "status = %0d\n", status);

    for (device = 0; device < nbDevices; device++) {
        status = FT_ListDevices((PVOID)((intptr_t)device), buffer, \
                               FT_LIST_BY_INDEX | FT_OPEN_BY_SERIAL_NUMBER);
        if (status == FT_OK) fprintf(stdout, "Driver ID: %s\n", buffer);
        else                  fprintf(stdout, "status = %0d\n", status);
    }
}
```

```
/*
 * \brief d2x_open
 *
 * - Open the device
 *
 * \param[in]      *driverID      Ptr on the driver ID "FTXG86D7D"
 * \param[out]     handle        Driver handle ID
 *
 */
FT_HANDLE d2x_open(const char_t *driverID, int32_t baudrate) {
    FT_STATUS status;
    FT_HANDLE handle;

    #if (defined(__APPLE__))
    system("sudo kextunload /System/Library/Extensions/FTDIUSBSerialDriver.kext/");
    status = FT_OpenEx((PVOID)driverID, FT_OPEN_BY_SERIAL_NUMBER, &handle);
    status = FT_SetBaudRate(handle, baudrate);
    system("sudo kextload /System/Library/Extensions/FTDIUSBSerialDriver.kext/");
    #elif (defined(__linux__))
    status = FT_OpenEx((PVOID)driverID, FT_OPEN_BY_SERIAL_NUMBER, &handle);
    status = FT_SetBaudRate(handle, baudrate);
    #else
    #error Platform not supported
    #endif

    return (handle);
}

/*
 * \brief d2x_close
 *
 * - Close the device
 *
 * \param[in]      handle        Driver handle ID
 * \param[out]     -
 *
 */
void d2x_close(FT_HANDLE handle) {
    FT_Close(handle);
}
```

```
/*
 * \brief d2x_readAByte
 *
 * - Read a byte on the device
 *
 * \param[in]      handle      Driver handle ID
 * \param[in]      *receive    Ptr on the receive buffer
 * \param[out]     status      FALSE -> OK
 *                          TRUE -> timeout error
 *
 */
bool_t d2x_readAByte(FT_HANDLE handle, uint8_t *receive) {
    FT_STATUS status;
    DWORD eventDWord;
    DWORD txBytes;
    DWORD rxBytes;
    DWORD bytesReceived;
    uint32_t timeout = 0;

    while (TRUE) {
        FT_GetStatus(handle, &rxBytes, &txBytes, &eventDWord);
        if (rxBytes > 0) break;
        usleep(10);
        if (timeout++ > 500000) return (TRUE);
    }

    status = FT_Read(handle, receive, 1, &bytesReceived);
    return (status == FALSE);
}
```

```
/*
 * \brief d2x_read
 *
 * - Read bytes on the device
 *
 * \param[in] handle Driver handle ID
 * \param[in] *receive Ptr on the receive buffer
 * \param[in] *count Ptr on the count
 * \param[out] status FALSE -> OK
 *              TRUE -> timeout error
 *
 */
bool_t d2x_read(FT_HANDLE handle, uint8_t *receive, uint32_t *count) {
    FT_STATUS status;
    DWORD eventDWord;
    DWORD txBytes;
    DWORD rxBytes;
    DWORD bytesReceived;
    uint32_t timeout = 0;

    while (TRUE) {
        FT_GetStatus(handle, &rxBytes, &txBytes, &eventDWord);
        if (rxBytes > 0) break;
        usleep(10);
        if (timeout++ > 500000) return (TRUE);
    }

    status = FT_Read(handle, receive, rxBytes, &bytesReceived);
    *count = (uint32_t)rxBytes;
    return (FALSE);
}
```

```
/*
 * \brief d2x_flush
 *
 * - Flush the device
 *
 * \param[in]      handle      Driver handle ID
 * \param[out]     -
 *
 */
void d2x_flush(FT_HANDLE handle) {
    uint8_t byte;
    DWORD eventDWord;
    DWORD txBytes;
    DWORD rxBytes;
    DWORD bytesReceived;

    while (TRUE) {
        FT_GetStatus(handle, &rxBytes, &txBytes, &eventDWord);
        if (rxBytes > 0) FT_Read(handle, &byte, 1, &bytesReceived);
        else return;
    }
}
```

5.1.2. The μKOS macros

The trace macro **TAG_LOG** can be used to generate data records (events) that will be collected in the log file. Of course this macro should be included in the program at the right place (avoiding to generate large quantity of data without significant information).

```
// Trace macros (for impulse)
// -----
#define TAG_LOG(commManager, id, s0,s1,s2,s3) { \
    volatile proc_t *process; \
    uint8_t log[64] = { \
        'u', 'K', 'O', 'S', /* + 0 */ 32, Header /* / \ */ \
        id, /* + 4 */ 8, Process ID /* / \ */ \
        0, /* + 5 */ 8, Process priority /* / \ */ \
        0,0, /* + 6 */ 16, Reserve (alignment) /* / \ */ \
        0,0,0,0, /* + 8 */ 32, Process Stack /* / \ */ \
        s0,s1,s2,s3, /* + 12 */ 32, Process State /* / \ */ \
        0,0,0,0,0,0,0, /* + 16 */ 64, Temporal tag us /* / \ */ \
        0,0,0,0,0,0,0, /* + 24 */ 64, Nb of executions /* / \ */ \
        0,0,0,0,0,0,0, /* + 32 */ 64, Cumulated time /* / \ */ \
    }; \
    kern_getProcessRun(&process); \
    kern_getTiccount((uint64_t *)(log + 16)); \
    *((uint8_t *)(log + 5)) = process->oSpecification.oPriority; \
    *((uint32_t *)(log + 8)) = (uint32_t)process->oSpecification.oStkSup; \
    *((uint64_t *)(log + 24)) = process->oStatistic.oNbExecutions; \
    *((uint64_t *)(log + 32)) = process->oStatistic.oTimePCum; \
    comm_write(commManager, (uint8_t *)&log[0], 40); \
}
};
```

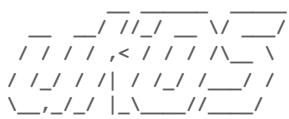
Usage examples:

```
TAG_LOG(commManager, id, s0,s1,s2,s3)

TAG_LOG(KUSBO, -1, 's','y','s',' ')
TAG_LOG(KUSBO, 0, 't','o','p',' ')
TAG_LOG(KUSBO, 3, 'h','e','l','p')
```

5.1.3. A complete example

```
/*
; misc_trace.
; =====
;
; -----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr          $: Author of last commit
; $Rev::: 35             $: Revision of last commit
; $Date::: 2014-01-25 11:52:31#$: Date of last commit
;
; Project:     uKOS
; Goal:        Demo of a C application.
;              This application shows how to operate with the uKOS uKernel.
;
;           On terminal type:
;           uKOSLog -d2x DCWZ1FL8 460800 /tmp/appl.log
;
;           Launch 3 processes:
;
;           Create the synchro semaphore "urt0 - RX char"
;           Configure the urt0 manager
;           Trace the program events
;
;           - P0: Every 1000-ms
;                 TRACE the process
;                 Toggle LED 0
;
;           - P1: Waiting for the semaphore "urt0 - RX char"
;                 TRACE the process
;                 Display a string or a timeout error message
;
;           - P2: Just use the CPU
;                 TRACE the process
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
; Franzi Edo.
; 5-Route de Cheseaux
; CH 1400 Cheseaux-Noréaz
; edo.franzi@ukos.ch
```



```
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include      <uKOS.h>
#include      <stdlib.h>

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "misc_trace   Example of how to use semaphores.           (c) EFr-2017";

LOC_CONST_STRG(aStrHelp[]) =
    "This is a ROMable C application\n"
    "=====\\n\\n"

    "This user function module is a C written application.\\n\\n"

    "Input format: misc_trace\\n"
    "Output format: [result]\\n\\n";
```

```
// Module specifications (See the modules.h)
// =====

MODULE(UserAppl, KIDAPPLICATION, KAPPLICATIONNUM, _start, "1.0", \
((1<<BSHOW) | (1<<BEXECONSOLE)));

/*
 * \brief Process 0
 *
 * - P0: - Every 1000-ms
 *       - Toggle LED 0
 *
 */
static void process_0(const void *argument) {

    while (TRUE) {
        TAG_LOG(KUSBO, -1, 'S','Y','S',' ')
        kern_suspendProcess(1000);
        TAG_LOG(KUSBO, 0, 't','o','p',' ')
        misc_toggleLed(0);
    }
}
```

```
/*
 * \brief Process 1
 *
 * - P1: - Waiting for the semaphore "urt0 - RX char"
 *       - Display a string or a timeout error message
 *
 */
static void process_1(const void *argument) {
    int32_t status;
    volatile sema_t *semaphore;

    kern_getSemaphoreById(KSEMAPHORE_URT0RX, &semaphore);

    while (TRUE) {
        TAG_LOG(KUSBO, -1, 'S','Y','S',' ')
        status = kern_waitSemaphore(semaphore, 2000);

        if (status == KKERNNTIMEO) {
            TAG_LOG(KUSBO, 1, 'e','r','r',' ')
            iotx_printf(KSYST, "Timeout Error process 1\n");
        }
        else {
            TAG_LOG(KUSBO, 1, 'o','k',' ', ' ')
            iotx_printf(KSYST, "KSYST - RX char\n");
        }
    }
}

/*
 * \brief Process 2
 *
 * - P2: - Every 10-ms
 *       - Just use the CPU
 *
 */
static void process_2(const void *argument) {

    while (TRUE) {
        TAG_LOG(KUSBO, -1, 'S','Y','S',' ')
        kern_suspendProcess(10);
        TAG_LOG(KUSBO, 2, 't','o','p',' ')
    }
}
```

```
/*
 * \brief main
 *
 * - Initialize the used libraries
 * - Launch all the processes
 * - Kill the "main". At this moment only the launched processes are executed
 *
 */
int    main(void) {
    cnfUrtx_t      configureURT0;
    volatile proc_t     *process_0, *process_10, *process_2;

    LOC_CONST_STRG(aStrIden_0[]) = "Process_User_0";
    LOC_CONST_STRG(aStrIden_1[]) = "Process_User_1";
    LOC_CONST_STRG(aStrIden_2[]) = "Process_User_2";
    LOC_CONST_STRG(aStrText_0[]) = "Process user 0.           (c) EFr-2017";
    LOC_CONST_STRG(aStrText_1[]) = "Process user 1.           (c) EFr-2017";
    LOC_CONST_STRG(aStrText_2[]) = "Process user 2.           (c) EFr-2017";

// Specifications for the processes

    PROC_SUPV(0, vSpecification_0, aStrText_0, KSZSTACKMIN, \
              process_0, aStrIden_0, KDEF0, 1, 0, KPROCNORMAL);
    PROC_SUPV(1, vSpecification_1, aStrText_1, KSZSTACKMIN, \
              process_1, aStrIden_1, KDEF0, 1, 0, KPROCNORMAL);
    PROC_SUPV(2, vSpecification_2, aStrText_2, KSZSTACKMIN, \
              process_2, aStrIden_2, KDEF0, 31, 0, KPROCNORMAL);

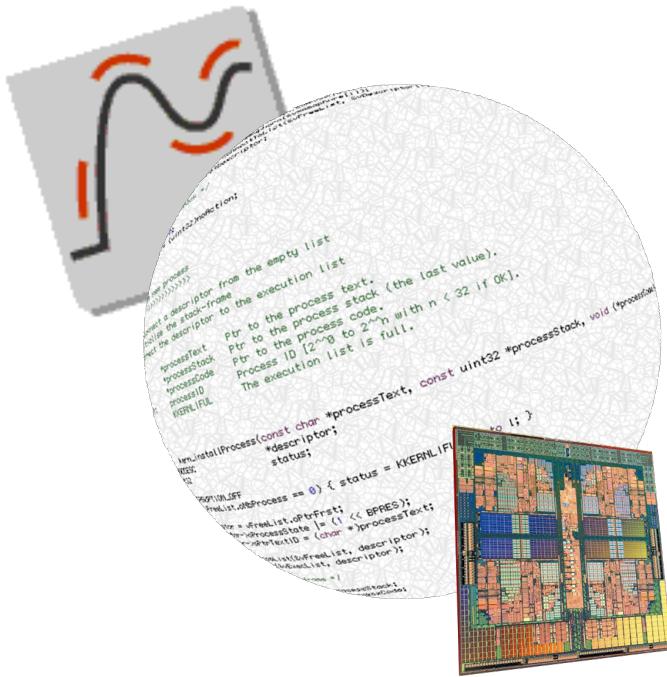
    configureURT0.oNBits      = K8BIT;
    configureURT0.oStopBits = K1STBIT;
    configureURT0.oParity   = KNONE;
    configureURT0.oBaudRate = KBDDEFAULT;
    configureURT0.oKernSync = (1<<BRXSEMA);
    urt0_configure(&configureURT0);

    if (kern_createProcess(&vSpecification_0, &process_0) != KKERNNOERR)
        exit(EXIT_FAILURE);
    if (kern_createProcess(&vSpecification_1, &process_1) != KKERNNOERR)
        exit(EXIT_FAILURE);
    if (kern_createProcess(&vSpecification_2, &process_2) != KKERNNOERR)
        exit(EXIT_FAILURE);
    return (EXIT_SUCCESS);
}
```

Here is the result after 1000000 events.



6. Sysquake Embedded



Sysquake documentation included with permission. Copyright 1997-2017, Calerga Sarl.

6.1. A new way to compute

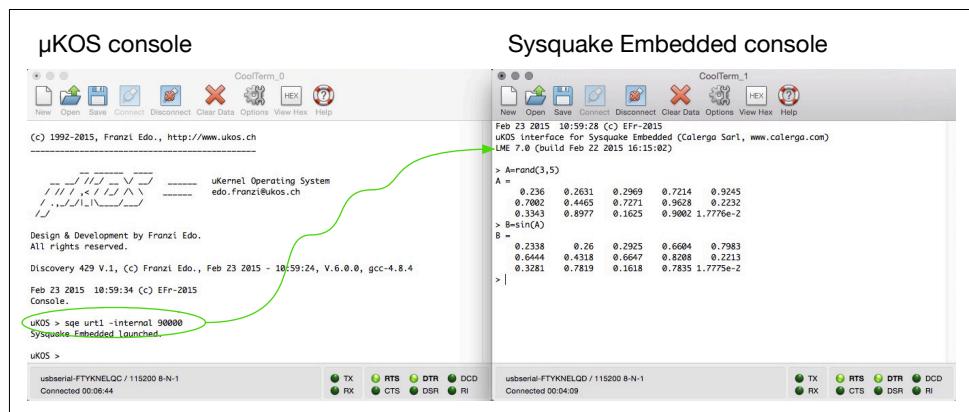
This chapter shows how to interact with the **Sysquake Embedded™ Engine (sqee)**. The idea is to launch in a separate process sqee and let μKOS to interact with it by a conventional command line or directly by another process.

6.1.1. Using sqee with a command line

This is the simplest way to operate. A μKOS tool named **sqe** is available for handle this functionality. In your serial terminal open a second window corresponding to the urt1 channel, then, from the μKOS console just type:

```
sqe urt1 -internal 90000
```

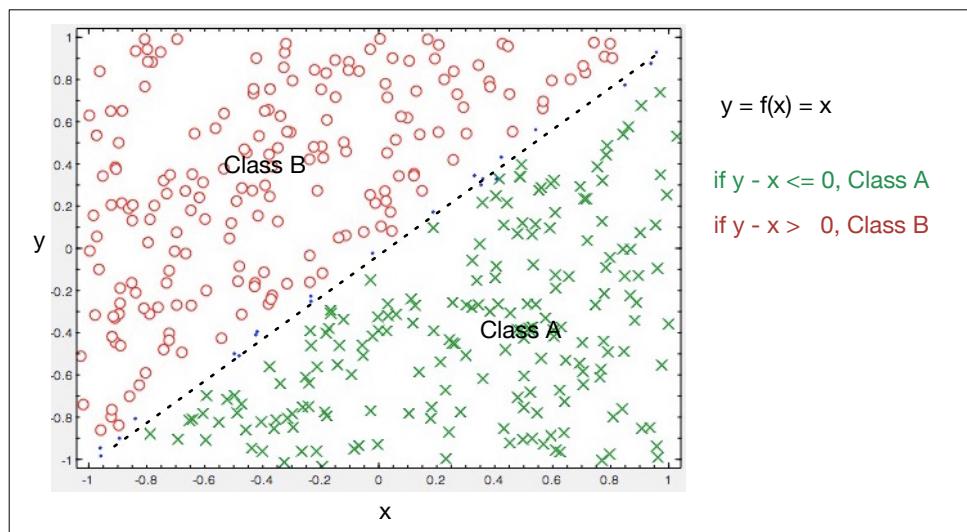
```
urt1      sqee console
-internal  Use the internal memory (usually the fastest one)
90000     Memory size allocated for sqee
```



6.1.2. Using sqee with a C program

This mode is for sure the most interesting. The interactions between **C** and **sqee** programs open new horizons (fast algorithm prototyping, complex applications using real data, etc.). The principle is simple: a **process** launches **sqee** and then it communicates with it via a **data structure**. In this way, C can now dispatch data and complex computing to the **sqee** engine.

The best way to emphasize this new way to mix the **languages C** and **LME** is to present a concrete application. The following program implements a feed forward 2 layers neural network. The training of the network was performed off line with the standard Sysquake package. The network was trained to classify a 2 component input vector {x-y} in 2 classes; the class A and the class B.



The C application performs these steps:

1. Launch in background the sqee process.
2. Load the LME network and the associated weight sets.
3. Ask sqee to generate a random input vector { -1 .. +1 } and fed the network.
4. Ask sqee for the results.

```
/*
; sqe_separe.
; =====

;-----  
; Author:      Franzi Edo.  The 2006-06-28  
; Modifs:  
;  
; SVN:  
; $Author:: efr           $: Author of last commit  
; $Rev:: 167             $: Revision of last commit  
; $Date:: 2014-12-31 15:29:49#$: Date of last commit  
;  
; Project:     uKOS  
; Goal:        Demo of a C application.  
;              This application shows how to operate with the uKOS uKernel.  
;  
;          Launch 1 processes:  
;  
;          - P0: Launch sqe  
;                  Every 1000-ms  
;                  Compute a NN (simple 2 classe separator)  
;  
; (c) 1992-2017, Franzi Edo.  
;-----  
;  
; Developed for use with Sysquake Embedded.  
; Sysquake is a trademark of Calerga Sarl, used with permission.  
;  
; Franzi Edo.          _ _ / / / \ \ /  
; 5-Route de Cheseaux   / / / , < / / / \ \ \ \ /  
; CH 1400 Cheseaux-Noréaz / / / / | / / / \ \ / /  
;                         \_,/_ / |_\ \ / / / \ /  
; edo.franzi@ukos.ch  
;  
; This program is free software: you can redistribute it and/or modify  
; it under the terms of the GNU Affero General Public License as published by  
; the Free Software Foundation, either version 3 of the License, or  
; (at your option) any later version.  
;  
; This program is distributed in the hope that it will be useful,  
; but WITHOUT ANY WARRANTY; without even the implied warranty of  
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
; GNU Affero General Public License for more details.  
;  
; You should have received a copy of the GNU Affero General Public License  
; along with this program. If not, see <http://www.gnu.org/licenses/>.  
;  
;-----  
*/
```

```
#include      <uKOS.h>

#define        KL1NBIN      2           // Number of inputs of the layer 1
#define        KL1NBOUT     5           // Number of neurons of the layer 1
#define        KL2NBIN      KL1NBOUT   // Number of inputs of the layer 2
#define        KL2NBOUT     2           // Number of outputs

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "sge_separe   Example of how to use sgee.                               (c) EFr-2017";

LOC_CONST_STRG(aStrHelp[]) =
    "This is a ROMable C application\n"
    "=====\\n\\n"

    "This user function module is a C written application.\\n\\n"

    "Input format:  sge_separe\\n"
    "Output format: [result]\\n\\n";

// Module specifications (See the modules.h)
// =====

MODULE(UserAppl, KIDAPPLICATION, KAPPLICATIONNUM, \
       _start, "1.0", ((1<<BSHOW) | (1<<BEXECONSOLE)))
```

```

/*
 * \brief aProcess_0
 *
 * - P0: - Launch the Sysquake Embedded Process
 *       - Every 1000-ms
 *       - Compute a NN (simple 2 class separator)
 *       - Toggle LED 0
 *
 */
static void aProcess_0(const void *argument) {
    uint32_t      size;
    uint8_t       *memory;
    float64_t     in[KL1NBIN];
    float64_t     out[KL2NBOUT];
    char_t        *res;
    cnfSqee_t    configure;
    ioSqee_t     data, *ref;
    const float64_t aL1_Weights[KL1NBOUT][KL1NBIN+1] = {
        { -0.8037,  0.5713, -0.1455 },
        { 0.7032, -0.2452,  0.1388 },
        { -2.0442,  2.1044, -4.4518e-2 },
        { 1.3805, -1.5087,  4.5242e-2 },
        { -1.7718,  1.837,  -4.6865e-2 }
    };
    const float64_t aL2_Weights[KL2NBOUT][KL2NBIN+1] = {
        { -0.8634,  0.3433, -1.8402,  1.2924, -2.1184, -0.1777 },
        { 0.4872, -0.6045,  2.3261, -1.6216,  1.3935,  0.1588 }
    };
}

// Try to reserve the SQE memory segment

size  = 90000;                                // 
memory = (uint8_t *)syos_malloc(KINTERNAL, size); // 
if (memory == 0) exit(EXIT_FAILURE);           // Reserve the memory

configure.oSize   = size;                      // Allocate 90'000-bytes
configure.oMemory = memory;                   // Address of the memory
sqee_configure((cnfSqee_t *)&configure);      // Configure the SQE

ref = &data;
ref->oSqeeRef = configure.oSqeeRef;          // Sqee handle

```

```
// Load the program

SQEE_COMPUTE(ref,
"define      KL1NBIN      = 2;\n"
"define      KL1NBOUT     = 5;\n"
"define      KL2NBIN      = KL1NBOUT;\n"
"define      KL2NBOUT     = 2;\n"

"L1_in          = zeros(KL1NBIN+1, 1);\n"
"L1_in(KL1NBIN+1, 1) = -1;\n"
"L1_activity    = zeros(KL1NBOUT, 1);\n"
"L1_out         = zeros(KL1NBOUT, 1);\n"
"L1_weight      = zeros(size(L1_out, 1), size(L1_in, 1));\n"

"L2_in          = zeros(KL2NBIN+1, 1);\n"
"L2_in(KL2NBIN+1, 1) = -1;\n"
"L2_activity    = zeros(KL2NBOUT, 1);\n"
"L2_out         = zeros(KL2NBOUT, 1);\n"
"L2_weight      = zeros(size(L2_out, 1), size(L2_in, 1));\n"

"function (input, output, activity) = _nn(input, weight, data);\n"
"  input(1:size(input, 1)-1, 1) = data;\n"
"  activity                  = weight * input;\n"
"  output                     = tanh(activity);");
);

// Load the weight set of the layer 1
// Load the weight set of the layer 2

SQEE_PUT(ref, K2DARRAY, 0, KL1NBIN+1, KL1NBOUT,
&aL1_Weights[0][0], "L1_weight = xdata;");
SQEE_PUT(ref, K2DARRAY, 0, KL2NBIN+1, KL2NBOUT,
&aL2_Weights[0][0], "L2_weight = xdata;");
```

```
while (TRUE) {
    kern_suspendProcess(1000);

// Generate a random input vector -1..+1
// Compute the NN

    SQEE_COMPUTE(ref,
    "data = (rand(2,1)-0.5)*2");

    SQEE_COMPUTE(ref,
    "(L1_in, L1_out, L1_activity) = _nn(L1_in, L1_weight, data);\n"
    "(L2_in, L2_out, L2_activity) = _nn(L2_in, L2_weight, L1_out);");

// Read the I/O of the NN

    SQEE_GET(ref, K1DARRAY, 0, 0, KL1NBIN, &in[0], "xdata(data);");
    SQEE_GET(ref, K1DARRAY, 0, 0, KL2NBOUT, &out[0], "xdata(L2_out);");

    if      ((out[0] > 0.2) && (out[1] < -0.2)) res = "Class A";
    else if ((out[0] < -0.2) && (out[1] > 0.2)) res = "Class B";
    else if ((out[0] > -0.2) && (out[1] < 0.2)) res = "Not well classed";
    else                                res = "Undetermined";

    iotx_printf(KSYST,
        "input x=%5d, y=%5d, result: (%5d %5d) %s\n",
        (int32_t)(in[0] * 1000),
        (int32_t)(in[1] * 1000),
        (int32_t)(out[0] * 1000),
        (int32_t)(out[1] * 1000),
        res);
}

}
```

```
/*
 * \brief main
 *
 * - Initialize the used libraries
 * - Launch all the processes
 * - Kill the "main". At this moment only the launched processes are executed
 *
 */
int    main(void) {
    volatile     proc_t      *process_0;

    LOC_CONST_STRG(aStrIden_0[]) =      "Process_User_0";
    LOC_CONST_STRG(aStrText_0[]) =      "User 0 process,           (c) EFr-2017";

// Specifications for the processes

    PROC_SUPV(0, vSpecification_0, aStrText_0, KSZSTACKMIN, aProcess_0, \
              aStrIden_0, KURT0, 1, 0, KPROCNORMAL);

    if (kern_createProcess(&vSpecification_0, &process_0) != KKERNNOERR)
        exit(EXIT_FAILURE);

    return (EXIT_SUCCESS_OS);
}
```

6.2. LME Tutorial

This section introduces **LME™** (Lightweight Math Engine), the interpreter for numerical computing used by **Sysquake**, and shows you how to perform basic computations. It supposes you can type commands to a command-line interface. You are invited to type the examples as you read this tutorial and to experiment on your own. For a more systematic description of **LME**, please consult the **Calerga** web site: http://www.calerga.com/doc/SQP_main.htm.

6.2.1. Simple operations

LME interprets what you type at the command prompt and displays the result unless you end the command with a semicolon. Simple expressions follow the syntactic rules of many programming languages.

```
> 2+3*4
ans =
14
> 2+3/4
ans =
2.75
```

As you can see, the evaluation order follows the usual rules which state that the multiplication (denoted with a **star**) and division (**slash**) have a higher priority than the addition and subtraction. You can change this order with parentheses:

```
> (2+3)*4
ans =
20
```

The result of expressions is automatically assigned to variable **ans** (more about variables later), which you can reuse in the next expression:

```
> 3*ans
ans =
60
```

Power is represented by the \wedge symbol:

```
> 2^5  
ans =  
32
```

LME has many mathematical functions. Trigonometric functions assume that angles are expressed in radians, and **sqrt** denotes the square root.

```
> sin(pi/4) * sqrt(2)  
ans =  
1
```

6.2.2. Complex Numbers

In many computer languages, the square root is defined only for nonnegative arguments. However, it is extremely useful to extend the set of numbers to remove this limitation. One defines **i** such that **i** * **i** = -1, and applies all the usual algebraic rules. For instance, **sqrt(-1)** = **sqrt(i*i)** = **i**, and **sqrt(-4)** = **sqrt(4)*sqrt(-1)** = **2i**. Complex numbers of the form **a + bi** are the sum of a real part **a** and an imaginary part **b**. It should be mentioned that **i**, the symbol used by mathematicians, is called **j** by engineers. **LME** accepts both symbols as input, but it always writes it **j**. You can use it like any function, or stick an **i** or **j** after a number:

```
> 2+3*j  
ans =  
2+3j  
> 3j+2  
ans =  
2+3j
```

Many functions accept complex numbers as argument, and return a complex result when the input requires it even if it is real:

```
> sqrt(-2)
ans =
0+1.4142i
> exp(3+2j)
ans =
-8.3585+18.2637j
> log(-8.3585+18.2637j)
ans =
3+2j
```

To get the real or imaginary part of a complex number, use the functions **real** or **imag**, respectively:

```
> real(2+3j)
ans =
2
> imag(2+3j)
ans =
3
```

Complex numbers can be seen as vectors in a plane. Then addition and subtraction of complex numbers correspond to the same operations applied to the vectors. The absolute value of a complex number, also called its magnitude, is the length of the vector:

```
> abs(3+4j)
ans =
5
> sqrt(3^2+4^2)
ans =
5
```

The argument of a complex number is the angle between the **x** axis ("real axis") and the vector, counterclockwise. It is calculated by the **angle** function.

```
> angle(2+3j)
ans =
0.9828
```

The last function specific to complex numbers we will mention here is **conj**, which calculates the conjugate of a complex number. The conjugate is simply the original number where the sign of the imaginary part is changed.

```
> conj(2+3j)
ans =
2-3j
```

Real numbers are also complex numbers, with a null imaginary part; hence

```
> abs(3)
ans =
3
> conj(3)
ans =
3
> angle(3)
ans =
0
> angle(-3)
ans =
3.1416
```

6.2.3. Vectors and Matrices

LME manipulates vectors and matrices as easily as scalars. To define a matrix, enclose its contents in **square** brackets and use commas to separate elements on the same row and semicolons to separate the rows themselves:

```
> [1,2;5,3]
ans =
1 2
5 3
```

Column vectors are matrices with one column, and row vectors are matrices with one row. You can also use the colon operator to build a row vector by specifying the start and end values, and optionally the step value. Note that the end value is included only if the range is a multiple of the step. Negative steps are allowed.

```
> 1:5
ans =
1 2 3 4 5
> 0:0.2:1
ans =
0 0.2 0.4 0.6 0.8 1
> 0:-0.3:1
ans =
0 -0.3 -0.6 -0.9
```

There are functions to create special matrices. The **zeros**, **ones**, **rand**, and **randn** functions create matrices full of zeros, ones, random numbers uniformly distributed between 0 and 1, and random numbers normally distributed with a mean of 0 and a standard deviation of 1, respectively. The eye function creates an identity matrix, i.e. a matrix with ones on the main diagonal and zeros elsewhere. All of these functions can take one scalar argument **n** to create a square **n-by-n** matrix, or two arguments **m** and **n** to create an **m-by-n** matrix.

```
> zeros(3)
ans =
0 0 0
0 0 0
0 0 0
> ones(2,3)
ans =
1 1 1
1 1 1
> rand(2)
ans =
0.1386 0.9274
0.3912 0.8219
> randn(2)
ans =
0.2931 1.2931
-2.3011 0.9841
> eye(3)
ans =
1 0 0
0 1 0
0 0 1
> eye(2,3)
ans =
1 0 0
0 1 0
```

You can use most scalar functions with matrices; functions are applied to each element.

```
> sin([1;2])
ans =
0.8415
0.9093
```

There are also functions which are specific to matrices. For example, **det** calculates the determinant of a square matrix:

```
> det([1,2;5,3])
ans =
-7
```

Arithmetic operations can also be applied to matrices, with their usual mathematical behavior. Additions and subtractions are performed on each element. The multiplication symbol * is used for the product of two matrices or a scalar and a matrix.

```
> [1,2;3,4] * [2;7]
ans =
16
34
```

The division symbol / denotes the multiplication by the inverse of the right argument (which must be a square matrix). To multiply by the inverse of the left argument, use the symbol \. This is handy to solve a set of linear equations. For example, to find the values of **x** and **y** such that **x+2y = 2** and **3 + 4y = 7**, type

```
> [1,2;3,4] \ [2;7]
ans =
3
-0.5
```

Hence $\mathbf{x} = \mathbf{3}$ and $\mathbf{y} = -0.5$. Another way to solve this problem is to use the **inv** function, which return the inverse of its argument. It is sometimes useful to multiply or divide matrices element-wise. The $\cdot*$, $\cdot/$ and $\cdot\wedge$ operators do exactly that. Note that the $+$ and $-$ operators do not need special dot versions, because they perform element-wise anyway.

```
> [1,2;3,4] * [2,1;5,3]
ans =
12 7
26 15
> [1,2;3,4] .* [2,1;5,3]
ans =
2 2
15 12
```

Some functions change the order of elements. The transpose operator (**tick**) reverses the columns and the rows:

```
> [1,2;3,4;5,6]'
ans =
1 3 5
2 4 6
```

When applied to complex matrices, the complex conjugate transpose is obtained. Use dot-tick if you just want to reverse the rows and columns. The **flipud** function flips a matrix upside-down, and **fliplr** flips a matrix left-right.

```
> flipud([1,2;3,4])
ans =
3 4
1 2
> fliplr([1,2;3,4])
ans =
2 1
4 3
```

To sort the elements of each column of a matrix, or the elements of a row vector, use the **sort** function:

```
> sort([2,4,8,7,1,3])
ans =
1 2 3 4 7 8
```

To get the size of a matrix, you can use the **size** function, which gives you both the number of rows and the number of columns unless you specify which of them you want in the optional second argument:

```
> size(rand(13,17))
ans =
13 17
> size(rand(13,17), 1)
ans =
13
> size(rand(13,17), 2)
ans =
17
```

6.2.4. Polynomials

LME handles mostly numerical values. Therefore, it cannot differentiate functions like $f(x) = \sin(\exp(x))$. However, a class of functions has a paramount importance in numerical computing, the polynomials. Polynomials are weighted sums of powers of a variable, such as $2x^2+3x-5$. LME stores the coefficients of polynomials in row vectors; i.e. $2x^2+3x-5$ is represented as [2,3,-5], and $2x^5+3x$ as [2,0,0,0,3,0].

Adding two polynomials would be like adding the coefficient vectors if they had the same size; in the general case, however, you had better use the function **addpol**, which can also be used for subtraction:

```
> addpol([1,2],[3,7])
ans =
 4 9
> addpol([1,2],[2,4,5])
ans =
 2 5 7
> addpol([1,2],[-2,4,5])
ans =
 -2 -3 -3
```

Multiplication of polynomials corresponds to convolution (no need to understand what it means here) of the coefficient vectors.

```
> conv([1,2],[2,4,5])
ans =
 2 8 13 10
```

Hence $(x+2)(2x^2+4x+5) = 2x^3+8x^2+13x+10$.

6.2.5. Strings

You type strings by delimiting them with **single quotes**:

```
> 'Hello, World!'
ans =
Hello, World!
```

If you want single quotes in a string, double them:

```
> 'Easy, isn''t it?'
ans =
Easy, isn't it?
```

Some control characters have a special representation. For example, the line feed, used in **LME** as an end-of-line character, is **\n**:

```
> 'Hello,\nWorld!'
ans =
Hello,
World!
```

Strings are actually matrices of characters. You can use commas and semicolons to build larger strings:

```
> ['a', 'bc', 'de', 'f']
ans =
abc
def
```

6.2.6. Variables

You can store the result of an expression into what is called a variable. You can have as many variables as you want and the memory permits. Each variable has a name to retrieve the value it contains. You can change the value of a variable as often as you want.

```
> a = 3;
> a + 5
ans =
8
> a = 4;
> a + 5
ans =
9
```

Note that a command terminated by a semicolon does not display its result. To see the result, remove the semicolon, or use a comma if you have several commands on the same line. Implicit assignment to variable ans is not performed when you assign to another variable or when you just display the contents of a variable.

```
> a = 3
a =
3
> a = 7, b = 3 + 2 * a
a =
7
b =
17
```

6.2.7. Loops and Conditional Execution

To repeat the execution of some commands, you can use either a **for/end** block or a **while/end** block. With for, you use a variable as a counter:

```
> for i=1:3;i,end
i =
1
i =
2
i =
3
```

With while, the commands are repeated as long as some expression is true:

```
> i = 1; while i < 10; i = 2 * i, end
i =
2
i =
4
i =
8
```

You can choose to execute some commands only if a condition holds true :

```
> if 2 < 3;'ok',else;'amazing...',end  
ans =  
ok
```

6.2.8. Functions

LME permits you to extend its set of functions with your own. This is convenient not only when you want to perform the same computation on different values, but also to make your code clearer by dividing the whole task in smaller blocks and giving names to them. To define a new function, you have to write its code in a file; you cannot do it from the command line. In **Sysquake**, put them in a function block.

Functions begin with a header which specifies its name, its input arguments (parameters which are provided by the calling expression) and its output arguments (result of the function). The input and output arguments are optional. The function header is followed by the code which is executed when the function is called. This code can use arguments like any other variables.

We will first define a function without any argument, which just displays a magic square, the sum of each line, and the sum of each column:

```
function magicsum3  
magic_3 = magic(3)  
sum_of_each_line = sum(magic_3, 2)  
sum_of_each_column = sum(magic_3, 1)
```

You can call the function just by typing its name in the command line:

```
> magicsum3  
magic_3 =  
8 1 6  
3 5 7  
4 9 2  
sum_of_each_line =  
15  
15  
15  
sum_of_each_column =  
15 15 15
```

This function is limited to a single size. For more generality, let us add an input argument:

```
function magicsum(n)
  magc = magic(n)
  sum_of_each_line = sum(magc, 2)
  sum_of_each_column = sum(magc, 1)
```

When you call this function, add an argument:

```
> magicsum(2)
magc =
  1 3
  4 2
sum_of_each_line =
  4
  6
sum_of_each_column =
  5 5
```

Note that since there is no **2-by-2** magic square, **magic(2)** gives something else... Finally, let us define a function which returns the sum of each line and the sum of each column:

```
function (sum_of_each_line, sum_of_each_column) = magicSum(n)
  magc = magic(n);
  sum_of_each_line = sum(magc, 2);
  sum_of_each_column = sum(magc, 1);
```

Since we can obtain the result by other means, we have added semicolons after each statement to suppress any output. Note the uppercase S in the function name: for **LME**, this function is different from the previous one. To retrieve the results, use the same syntax:

```
> (sl, sc) = magicSum(3)
sl =
  15
  15
  15
sc =
  15 15 15
```

You do not have to retrieve all the output arguments. To get only the first one, just type

```
> s1 = magicSum(3)
s1 =
 15
 15
 15
```

When you retrieve only one output argument, you can use it directly in an expression:

```
> magicSum(3) + 3
ans =
 18
 18
 18
```

One of the important benefits of defining function is that the variables have a limited scope. Using a variable inside the function does not make it available from the outside; thus, you can use common names (such as **x** and **y**) without worrying about whether they are used in some other part of your whole program. For instance, let us use one of the variables of **magicSum**:

```
> magc = 77
magc =
 77
> magicSum(3) + magc
ans =
 92
 92
 92
> magc
magc =
 77
```

6.2.9. Local and Global Variables

When a value is assigned to a variable which has never been referenced, a new variable is created. It is visible only in the current context: the base workspace for assignments made from the command-line interface, or the current function invocation for functions. The variable is discarded when the function returns to its caller. Variables can also be declared to be global, i.e. to survive the end of the

function and to support sharing among several functions and the base workspace. Global variables are declared with keyword **global**:

```
global x  
global y z
```

A global variable is unique if its name is unique, even if it is declared in several functions.

In the following example, we define functions which implement a queue which contains scalar numbers. The queue is stored in a global variable named **QUEUE**. Elements are added at the front of the vector with function **queueput**, and retrieved from the end of the vector with function **queueget**.

```
function queueput(x)  
    global QUEUE;  
    QUEUE = [x, QUEUE];  
  
function x = queueget  
    global QUEUE;  
    x = QUEUE(end);  
    QUEUE(end) = [];
```

Both functions must declare **QUEUE** as global; otherwise, the variable would be local, even if there exists also a global variable defined elsewhere. The first time a global variable is defined, its value is set to the empty matrix **[]**. In our case, there is no need to initialize it to another value.

Here is how these functions can be used.

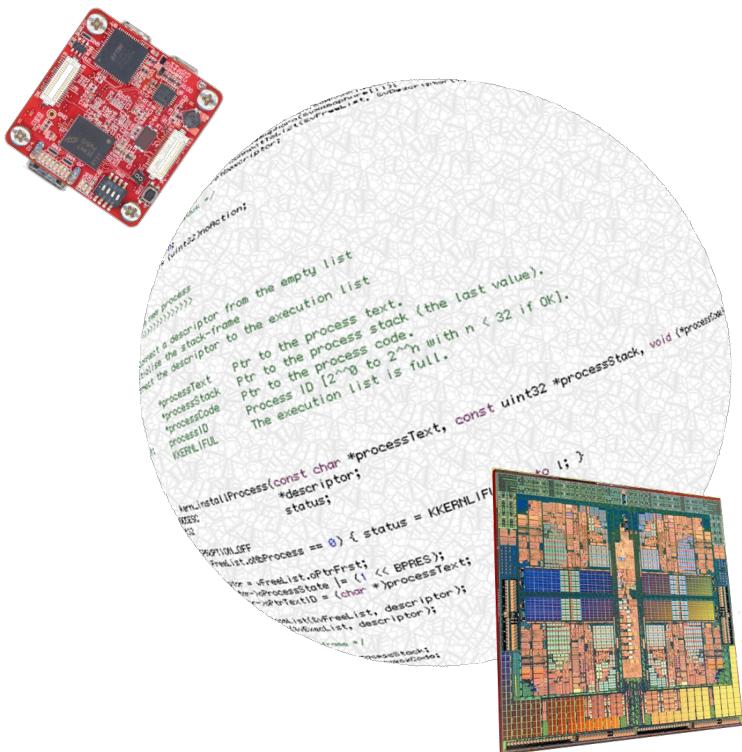
```
> queueput(1);  
> queueget  
ans =  
     1  
> queueput(123);  
> queueput(2+3j);  
> queueget  
ans =  
     123  
> queueget  
ans =  
     2 + 3j
```

To observe the value of **QUEUE** from the command-line interface, **QUEUE** must be declared global there. If a local variable **QUEUE** already exists, it is discarded.

```
> global QUEUE
> QUEUE
QUEUE =
[ ]
> queueput(25);
> queueput(17);
> QUEUE
QUEUE =
17 25
```

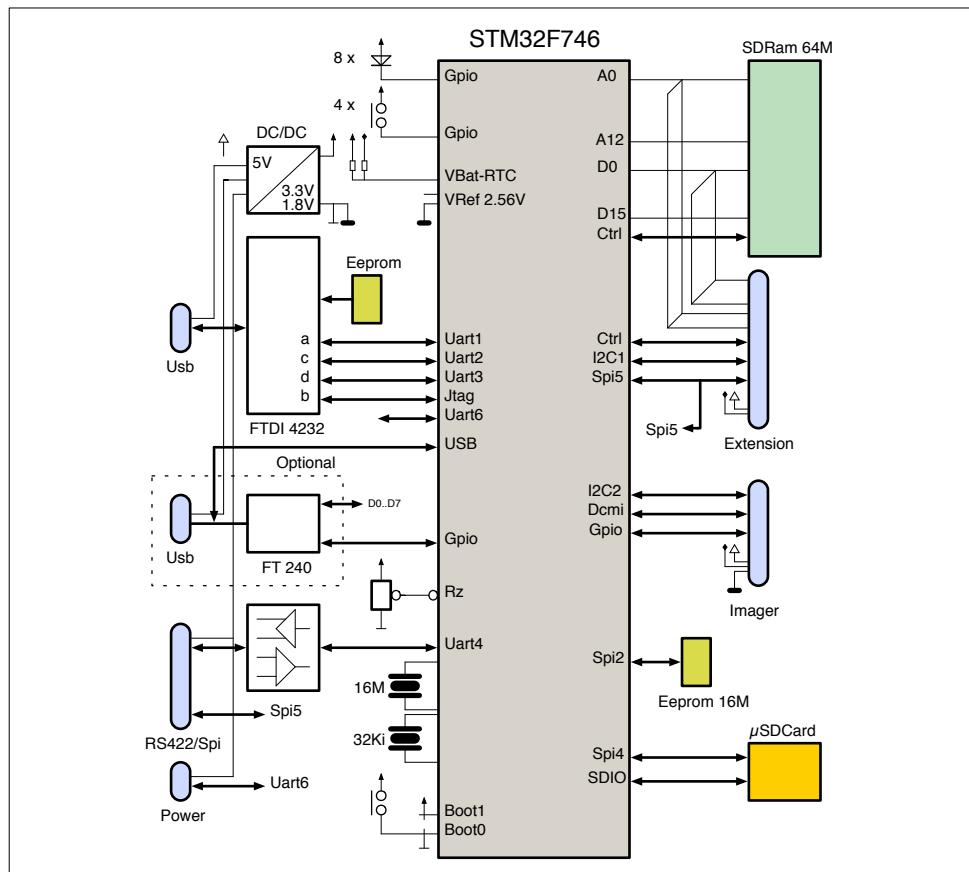
v 4.8

7. The cortex-M7F



7.1. The cortex-M7F board

- CPU ARM cortex-M7F 216-MHz μController.
- 64-MBytes of SDRAM (32-M x 16-bits).
- 2-MBytes of SPI EEPROM.
- Extension bus (I/O, CPU, imager).
- 3 serial interfaces + JTAG (via FTDI).
- 1 FIFO interface (via FTDI).
- USB 1.1.
- 8 LEDs & Configuration HEXA switches.
- DC/DC on board (Power-in up to 5.5-V).
- 1 serial interface with a RS422.
- 1 serial interface with a possible Bluetooth.
- μSDCard interface.



7.2. The CPU unit

The **STM32F746** μController is based on the high-performance **ARM cortex-M7F** 32-bit RISC core operating at a frequency of up to 216-MHz. The cortex-M7F core features a Floating Point Unit (**FPU**) single/double precision which supports all ARM single- double- precision data-processing instructions and data types. It also implements a full set of **DSP** instructions and a memory protection unit (MPU) which enhances application security.

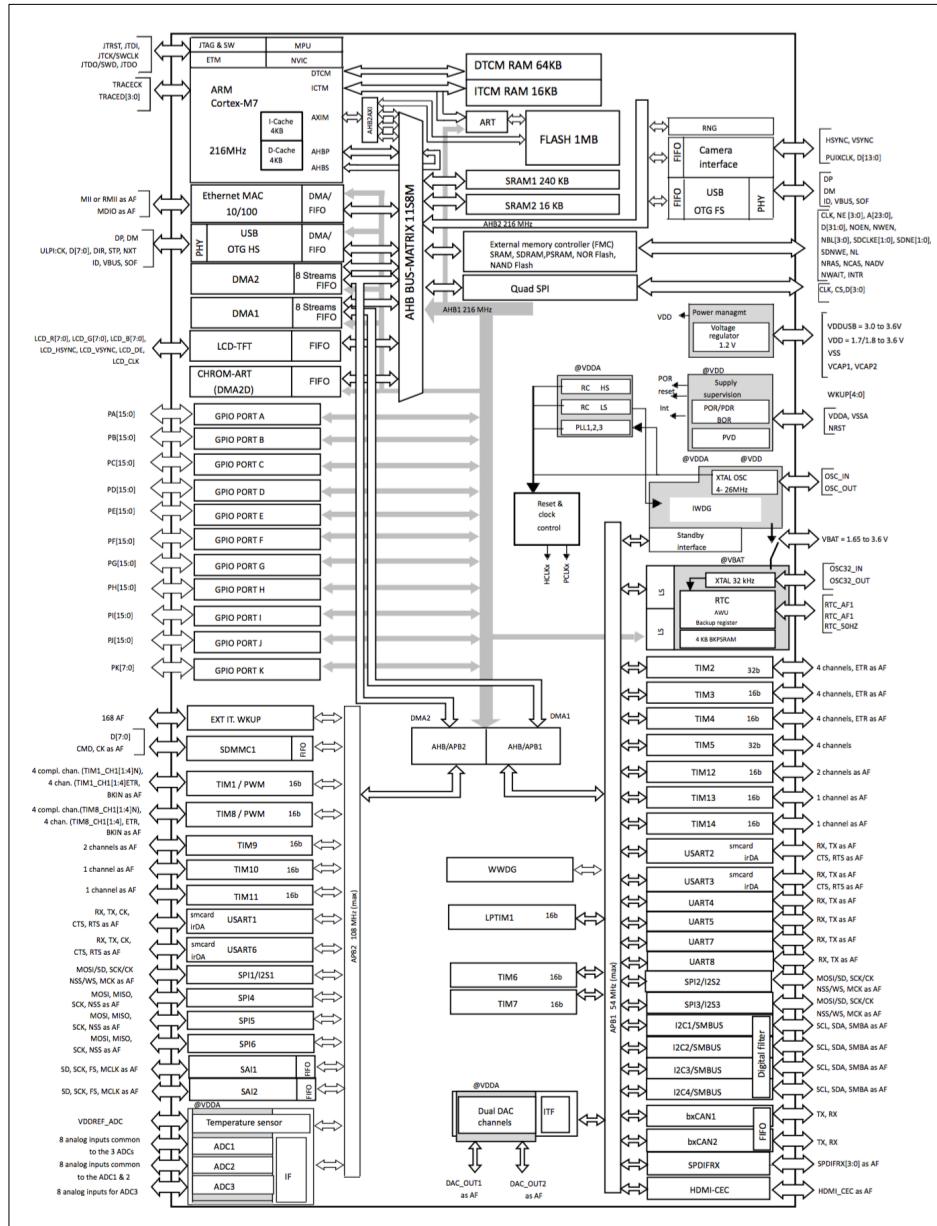
The **STM32F746** μController incorporates high-speed embedded memories (1-Mbyte of Flash and 320-Kbytes of SRAM), and an extensive range of enhanced I/Os and peripherals connected to two **APB** buses, two **AHB** buses and a 32-bit multi-**AHB** bus matrix.

All devices offer three 12-bit **ADCs**, two **DACs**, a low-power **RTC**, twelve general-purpose 16-bit timers including two **PWM**, two general-purpose 32-bit timers and a true random number generator **RNG**. It also feature standard and advanced communication interfaces.

Here are some of them:

- Up to three I2Cs.
- Six SPIs, two I2Ss full duplex. To achieve audio class accuracy, the I2S peripherals can be clocked via a dedicated internal audio PLL or via an external clock to allow synchronization.
- Four USARTs plus four UARTs.
- An USB OTG full-speed and a USB OTG high-speed with full-speed capability (with the ULPI).
- Two CANs.
- An SDIO/MMC interface.
- Ethernet and the camera interface.

New advanced peripherals include an **SDIO**, an enhanced flexible static memory control (**FMC**) interface and a camera interface (**DCMI**) for CMOS sensors. The **STM32F746** is a complex μController, and its peripherals allow to reduce dramatically the BOM of a system. The bloc schematics of the device is shown below.



7.3. Pins / interfaces configuration

The μController includes a Pin-to-Function interconnection matrix. This facilitates system development and the PCB routing. Almost every pin can be used as a generic GPIO or can be redirected to a specific peripheral signal. The pins can be configured to have pull-ups or pull-down, to be fast or slow, to be digital or analog, etc.

For the **Baphomet-746** here is the selected signals and their attachment to the interfaces.

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SA1	SPI2/3/U SART1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/UA RT4/5/7/8 /SPDIFR X	CAN1/2/T M1/2/13/ 14/QUAD SPI/LCD	SAI2/QU ADSPPIO TG2_HS/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port A	PA0	-	TIM2_C H1/TIM2 _ETR	TIM5_C H1	TIM8_ET R	-	-	-	USART2 _CTS	UART4_ TX	-	SAI2_SD_ B	ETH_MII_ CRS	-	-	-	EVEN TOUT
	PA1	-	TIM2_C H2	TIM5_C H2	-	-	-	-	USART2 _RTS	UART4_ RX	QUADSP L_BK1_IO 3	SAI2_MC K_B	ETH_MII/ RX_CLK/ ETH_RMI/ L_REF_C LK	-	-	LCD_R2	EVEN TOUT
	PA2	-	TIM2_C H3	TIM5_C H3	TIM9_CH 1	-	-	-	USART2 _TX	SAI2_SC K_B	-	-	ETH_MDI O	-	-	LCD_R1	EVEN TOUT
	PA3	-	TIM2_C H4	TIM5_C H4	TIM9_CH 2	-	-	-	USART2 _RX	-	-	OTG_HS_ ULPI_D0	ETH_MII_ COL	-	-	LCD_B5	EVEN TOUT
	PA4	-	-	-	-	-	SPI1_NS S/I2S1_ WS	SPI3_NS S/I2S3_ WS	USART2 _CK	-	-	-	OTG_HS_ SOF	DCMI_H SYNC	LCD_VS YNC	EVEN TOUT	
	PA5	-	TIM2_C H1/TIM2 _ETR	-	TIM8_CH 1N	-	SPI1_SC K/I2S1_ CK	-	-	-	-	OTG_HS_ ULPI_CK	-	-	LCD_R4	EVEN TOUT	
	PA6	-	TIM1_B KIN	TIM3_C H1	TIM8_BKI N	-	SPI1_MI SO	-	-	TIM13_C H1	-	-	DCMI_PI XCLK	LCD_G2	-	EVEN TOUT	
	PA7	-	TIM1_C H1N	TIM3_C H2	TIM8_CH 1N	-	SPI1_M OS/I2S1 _SD	-	-	TIM14_C H1	-	ETH_MII/ RX_DV/E TH_RMI/ CRS_DV	FMC_SD NWE	-	-	EVEN TOUT	
	PA8	MCO1	TIM1_C H1	-	TIM8_BKI N2	I2C3_SC L	-	-	USART1 _CK	-	-	OTG_FS_ SOF	-	-	LCD_R6	EVEN TOUT	
	PA9	-	TIM1_C H2	-	-	I2C3_SM BA	SPI2_SC K/I2S2 CK	-	USART1 _TX	-	-	-	-	DCMI_D 0	-	EVEN TOUT	
	PA10	-	TIM1_C H3	-	-	-	-	-	USART1 _RX	-	-	OTG_FS_ ID	-	-	DCMI_D 1	-	EVEN TOUT
	PA11	-	TIM1_C H4	-	-	-	-	-	USART1 _CTS	-	CAN1_R X	OTG_FS_ DM	-	-	-	LCD_R4	EVEN TOUT

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SA1	SPI2/3/U SART1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/UA RT4/5/7/8 /SPDIFR X	CAN1/2/T M12/13/ 14/QUAD SPI/LCD	SAI2/QU ADSPPIO TG2_H/S/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port A	PA12	-	TIM1_ET_R	-	-	-	-	-	USART1_RTS	SAI2_FS_B	CAN1_T_X	OTG_FS_DP	-	-	-	LCD_R5	EVEN TOUT
	PA13	JTMS_SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT	
	PA14	JTCK_SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT	
	PA15	JTDI	TIM2_C_H1/TIM2_ETR	-	-	HDMI-CEC	SPI1_NS_S/I2S1_WS	SPI3_NS_S/I2S3_WS	-	UART4_RTS	-	-	-	-	-	EVEN TOUT	
Port B	PB0	-	TIM1_C_H2N	TIM3_C_H3	TIM8_CH_2N	-	-	-	UART4_CTS	LCD_R3	OTG_HS_ULPI_D1	ETH_MII_RXD2	-	-	-	EVEN TOUT	
	PB1	-	TIM1_C_H3N	TIM3_C_H4	TIM8_CH_3N	-	-	-	-	LCD_R6	OTG_HS_ULPI_D2	ETH_MII_RXD3	-	-	-	EVEN TOUT	
	PB2	-	-	-	-	-	-	SAI1_SD_A	SPI3_MO_S/I2S3_SD	QUADSP_I_CLK	-	-	-	-	-	EVEN TOUT	
	PB3	JTDO/T_RACES_WO	TIM2_C_H2	-	-	-	SPI1_SC_K/I2S1_CK	SPI3_SC_K/I2S3_CK	-	-	-	-	-	-	-	EVEN TOUT	
	PB4	NJTRST	-	TIM3_C_H1	-	-	SPI1_MI_SO	SPI3_MI_SO	SPI2_NS_S/I2S2_WS	-	-	-	-	-	-	EVEN TOUT	
	PB5	-	-	TIM3_C_H2	-	I2C1_SM_BA	SPI1_M_OSI/I2S1_SO	SPI3_M_OSI/I2S3_SO	-	CAN2_R_X	OTG_HS_ULPI_D7	ETH_PPS_OUT	FMC_SD_CKE1	DCMI_D_10	-	EVEN TOUT	
	PB6	-	-	TIM4_C_H1	HDMI-CEC	I2C1_SC_L	-	-	USART1_TX	-	CAN2_T_X	QUADSPI_BK1_NC_S	-	FMC_SD_NE1	DCMI_D_5	-	EVEN TOUT
	PB7	-	-	TIM4_C_H2	-	I2C1_SD_A	-	-	USART1_RX	-	-	-	-	FMC_NL	DCMI_V_SYNC	-	EVEN TOUT
	PB8	-	-	TIM4_C_H3	TIM10_C_H1	I2C1_SC_L	-	-	-	CAN1_R_X	-	ETH_MII_TXD3	SDMMC_1_D4	DCMI_D_6	LCD_B6	EVEN TOUT	

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SA1	SPI2/3/U SART1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/UA RT4/5/7/8 /SPDIFR X	CAN1/2/T M12/13/ 14/QUAD SPI/LCD	SAI2/QU ADSPPIO TG2_H/S/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port B	PB9	-	-	TIM4_C_H4	TIM11_CH_1	I2C1_SD_A	SPI2_NS_S/I2S2_WS	-	-	-	CAN1_T_X	-	-	SDMMC_1_D5	DCMI_D_7	LCD_B7	EVEN TOUT
	PB10	-	TIM2_C_H3	-	-	I2C2_SC_L	SPI2_SC_K/I2S2_CK	-	USART3_TX	-	-	OTG_HS_ULPI_D3	ETH_MII_RX_ER	-	-	LCD_G4	EVEN TOUT
	PB11	-	TIM2_C_H4	-	-	I2C2_SD_A	-	-	USART3_RX	-	-	OTG_HS_ULPI_D4	ETH_MII_TX_EN/ETH_RMII_TX_EN	-	-	LCD_G5	EVEN TOUT
	PB12	-	TIM1_B_KIN	-	-	I2C2_SM_BA	SPI2_NS_S/I2S2_WS	-	USART3_CK	-	CAN2_R_X	OTG_HS_ULPI_D5	ETH_MII_RX_ER/H_RMII_T_XD1	OTG_HS_ID	-	EVEN TOUT	
	PB13	-	TIM1_C_H1N	-	-	-	SPI2_SC_K/I2S2_CK	-	USART3_CTS	-	CAN2_T_X	OTG_HS_ULPI_D6	ETH_MII_RXD1/ETH_RMII_T_XD1	-	-	EVEN TOUT	
	PB14	-	TIM1_C_H2N	-	TIM8_CH_2N	-	SPI2_MI_SO	-	USART3_RTS	-	TIM12_C_H1	-	-	OTG_HS_DM	-	EVEN TOUT	
	PB15	RTC_R_EFIN	TIM1_C_H3N	-	TIM8_CH_3N	-	SPI2_M_OSI/I2S2_SD	-	-	-	TIM12_C_H2	-	-	OTG_HS_DP	-	EVEN TOUT	
Port C	PC0	-	-	-	-	-	-	-	-	SAI2_FS_B	-	OTG_HS_ULPI_STP	-	FMC_SD_NWE	-	LCD_R5	EVEN TOUT
	PC1	TRACE_D0	-	-	-	-	SPI2_M_OSI/I2S2_SD	SPI1_SD_A	-	-	-	-	ETH_MD_C	-	-	EVEN TOUT	
	PC2	-	-	-	-	-	SPI2_MI_SO	-	-	-	-	OTG_HS_DIR	ETH_MII_TXD2	FMC_SD_NE0	-	EVEN TOUT	
	PC3	-	-	-	-	-	SPI2_M_OSI/I2S2_SD	-	-	-	-	OTG_HS_ULPI_NT	ETH_MII_TX_CLK	FMC_SD_CKE0	-	EVEN TOUT	

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15	
	SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SP1/2/3/ 4/5/6	SPI3/ SAI1	SPI2/3/U SAI1/T1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/U RT4/5/7/8 /SPDIFRX	CAN1/2/T IM1/2/3/ 14/QUAD SPI/LCD	SAI2/OU ADSP/IO TG2_HS/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS	
Port C	PC4	-	-	-	-	-	I2S1_M CK	-	-	SPDIFRX _IN2	-	-	ETH_MII RXD0/ET H_RMI_RXD0	FMC_SD NE0	-	-	EVEN TOUT
	PC5	-	-	-	-	-	-	-	-	SPDIFRX _IN3	-	-	ETH_MII RXD1/ET H_RMI_RXD1	FMC_SD CKE0	-	-	EVEN TOUT
	PC6	-	-	TIM3_C H1	TIM8_CH 1	-	I2S2_M CK	-	USART6 _TX	USART6 _RX	-	-	SDMMC _1_D6	DCMI_D 0	LCD_HS YNC	EVEN TOUT	
	PC7	-	-	TIM3_C H2	TIM8_CH 2	-	-	I2S3_M CK	-	USART6 _RX	-	-	SDMMC _1_D7	DCMI_D 1	LCD_G6	EVEN TOUT	
	PC8	TRACE D1	-	TIM3_C H3	TIM8_CH 3	-	-	UART5_S _RTS	UART5_T _CK	-	-	-	SDMMC _1_D0	DCMI_D 2	-	EVEN TOUT	
	PC9	MCO2	-	TIM3_C H4	TIM8_CH 4	I2C3_SD A	I2S_N CK	-	UART5_S _CTS	-	QUADSP _1_BK1_IO 0	-	-	SDMMC _1_D1	DCMI_D 3	-	EVEN TOUT
	PC10	-	-	-	-	-	-	SPI3_SC K/I2S3_ CK	USART3 _TX	UART4_T _X	QUADSP _1_BK1_IO 1	-	-	SDMMC _1_D2	DCMI_D 8	LCD_R2	EVEN TOUT
	PC11	-	-	-	-	-	-	SPI3_MI SO	USART3 _RX	UART4_T _RX	QUADSP _1_BK2_N CS	-	-	SDMMC _1_D3	DCMI_D 4	-	EVEN TOUT
	PC12	TRACE D3	-	-	-	-	-	SPI3_M OS/I2S3_ SD	USART3_S _CK	UART5_T _X	-	-	-	SDMMC _1_CK	DCMI_D 9	-	EVEN TOUT
	PC13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT
	PC14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT
	PC15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15	
	SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SP1/1/2/3/ 4/5/6	SPI3/ SAI1	SPI2/3/U SAI1/T1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/U RT4/5/7/8 /SPDIFRX	CAN1/2/T IM1/2/3/ 14/QUAD SPI/LCD	SAI2/OU ADSP/IO TG2_HS/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS	
Port D	PD0	-	-	-	-	-	-	-	-	CAN1_R X	-	-	FMC_D2	-	-	EVEN TOUT	
	PD1	-	-	-	-	-	-	-	-	CAN1_T X	-	-	FMC_D3	-	-	EVEN TOUT	
	PD2	TRACE D2	-	TIM3_ET R	-	-	-	-	UART5_S _RX	-	-	-	SDMMC _1_CMD	DCMI_D 11	-	EVEN TOUT	
	PD3	-	-	-	-	-	-	SPI2_SC K/I2S2_ CK	-	USART2_S _CTS	-	-	-	FMC_CL K	DCMI_D 5	LCD_G7	EVEN TOUT
	PD4	-	-	-	-	-	-	-	USART2_S _RTS	-	-	-	-	FMC_N OE	-	-	EVEN TOUT
	PD5	-	-	-	-	-	-	-	USART2_S _TX	-	-	-	-	FMC_N WE	-	-	EVEN TOUT
	PD6	-	-	-	-	-	-	SPI3_M OS/I2S3_ SD	SPI1_SD _A	USART2_S _RX	-	-	-	FMC_N WAIT	DCMI_D 10	LCD_B2	EVEN TOUT
	PD7	-	-	-	-	-	-	-	USART2_S _CK	SPDIFRX _IN0	-	-	-	FMC_NE 1	-	-	EVEN TOUT
	PD8	-	-	-	-	-	-	-	USART3_S _TX	SPDIFRX _IN1	-	-	-	FMC_D1 3	-	-	EVEN TOUT
	PD9	-	-	-	-	-	-	-	USART3_S _RX	-	-	-	-	FMC_D1 4	-	-	EVEN TOUT
	PD10	-	-	-	-	-	-	-	USART3_S _CK	-	-	-	-	FMC_D1 5	-	LCD_B3	EVEN TOUT
	PD11	-	-	-	-	I2C4_SM BA	-	-	USART3_S _CTS	-	QUADSP _1_BK1_IO 0	SPI2_SD _A	-	FMC_A1 6/FMC_CLE	-	-	EVEN TOUT
	PD12	-	-	TIM4_C H1	LPTIM1_I N1	I2C4_SC L	-	-	USART3_S _RTS	-	QUADSP _1_BK1_IO 1	SPI2_FS _A	-	FMC_A1 7/FMC_ALE	-	-	EVEN TOUT
	PD13	-	-	TIM4_C H2	LPTIM1_O UT	I2C4_SD A	-	-	-	-	QUADSP _1_BK1_IO 3	SPI2_SC K_A	-	FMC_A1 8	-	-	EVEN TOUT

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SP1/2/3/ 4/SI6	SP13/ SA11	SP12/3/U SART1/2/ 3/UART5/ SPDIFRX	SA12/US ART6/UA RT4/5/7/8 /SPDIF/RX	CAN1/2/T IM12/13/ 14/QUAD SI/PLCD	SA12/QU ADSP/IO TG2_H/S OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port D	PD14	-	-	TIM4_C H3	-	-	-	-	UART8_CTS	-	-	-	FMC_D0	-	-	EVEN TOUT	
	PD15	-	-	TIM4_C H4	-	-	-	-	UART8_RTS	-	-	-	FMC_D1	-	-	EVEN TOUT	
Port E	PE0	-	-	TIM4_ET R	LPTIM1_E TR	-	-	-	UART8_Rx	-	SA12_MC K_A	-	FMC_NB L0	DCMI_D 2	-	EVEN TOUT	
	PE1	-	-	-	LPTIM1_I N2	-	-	-	UART8_T_x	-	-	-	FMC_NB L1	DCMI_D 3	-	EVEN TOUT	
	PE2	TRACE_CLK	-	-	-	-	SP14_SC K	SA11_M CLK_A	-	-	QUADSP I_BK1_IO 2	-	ETH_MII TXD3	FMC_A2 3	-	EVEN TOUT	
	PE3	TRACE_D0	-	-	-	-	-	SA11_SD B	-	-	-	-	FMC_A1 9	-	-	EVEN TOUT	
	PE4	TRACE_D1	-	-	-	-	SP14_NS SO	SA11_FS A	-	-	-	-	FMC_A2 0	DCMI_D 4	LCD_B0	EVEN TOUT	
	PE5	TRACE_D2	-	-	TIM9_CH 1	-	SP14_MI SO	SA11_SC K_A	-	-	-	-	FMC_A2	DCMI_D 1	LCD_G0	EVEN TOUT	
	PE6	TRACE_D3	TIM1_B KIN2	-	TIM9_CH 2	-	SP14_M OSI	SA11_SD A	-	-	-	SA12_MC K_B	FMC_A2 2	DCMI_D 7	LCD_G1	EVEN TOUT	
	PE7	-	TIM1_ET R	-	-	-	-	-	UART7_Rx	-	QUADSP I_BK2_IO0	-	FMC_D4	-	-	EVEN TOUT	
	PE8	-	TIM1_C H1/N	-	-	-	-	-	UART7_T_x	-	QUADSP I_BK2_IO1	-	FMC_D5	-	-	EVEN TOUT	
	PE9	-	TIM1_C H1	-	-	-	-	-	UART7_RTS	-	QUADSP I_BK2_IO2	-	FMC_D6	-	-	EVEN TOUT	
	PE10	-	TIM1_C H2/N	-	-	-	-	-	UART7_CTS	-	QUADSP I_BK2_IO3	-	FMC_D7	-	-	EVEN TOUT	
	PE11	-	TIM1_C H2	-	-	-	SP14_NS S	-	-	-	SA12_SD B	-	FMC_D8	-	LCD_G3	EVEN TOUT	
	PE12	-	TIM1_C H3/N	-	-	-	SP14_SC K	-	-	-	SA12_SC K_B	-	FMC_D9	-	LCD_B4	EVEN TOUT	
	PE13	-	TIM1_C H3	-	-	-	SP14_MI SO	-	-	-	SA12_FS B	-	FMC_D1 0	-	LCD_DE	EVEN TOUT	

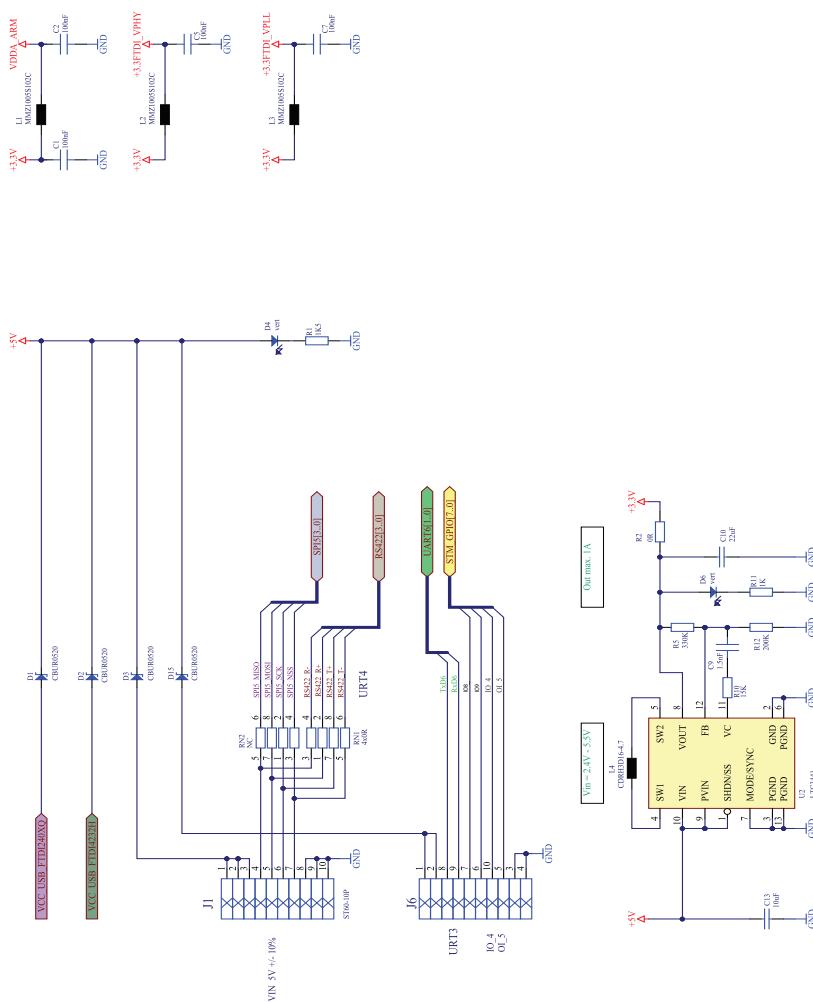
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SP1/2/3/ 4/SI6	SP13/ SA11	SP12/3/U SART1/2/ 3/UART5/ SPDIFRX	SA12/US ART6/UA RT4/5/7/8 /SPDIF/RX	CAN1/2/T IM12/13/ 14/QUAD SI/PLCD	SA12/QU ADSP/IO TG2_H/S OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port F	PE14	-	TIM1_C H4	-	-	-	SP14_M OSI	-	-	-	SA12_MC K_B	-	FMC_D1 1	-	LCD_CL K	EVEN TOUT	
	PE15	-	TIM1_B KIN	-	-	-	-	-	-	-	-	-	FMC_D1 2	-	LCD_R7	EVEN TOUT	
	PF0	-	-	-	-	I2C2_SD A	-	-	-	-	-	-	FMC_A0	-	-	EVEN TOUT	
	PF1	-	-	-	-	I2C2_SC L	-	-	-	-	-	-	FMC_A1	-	-	EVEN TOUT	
	PF2	-	-	-	-	I2C2_SM BA	-	-	-	-	-	-	FMC_A2	-	-	EVEN TOUT	
	PF3	-	-	-	-	-	-	-	-	-	-	-	FMC_A3	-	-	EVEN TOUT	
	PF4	-	-	-	-	-	-	-	-	-	-	-	FMC_A4	-	-	EVEN TOUT	
	PF5	-	-	-	-	-	-	-	-	-	-	-	FMC_A5	-	-	EVEN TOUT	
	PF6	-	-	TIM10_C H1	-	SP15_NS S	SA11_SD B	-	UART7_RX	QUADSP I_BK1_IO 3	-	-	-	-	-	EVEN TOUT	
	PF7	-	-	-	TIM11_CH 1	-	SP15_SC K	SA11_M CLK_B	-	UART7_T_x	QUADSP I_BK1_IO 2	-	-	-	-	EVEN TOUT	
	PF8	-	-	-	-	-	SP15_MI SO	SA11_SC K_B	-	UART7_RTS	TIM13_C H1	QUADSP I_BK1_IO0	-	-	-	EVEN TOUT	
	PF9	-	-	-	-	-	SP15_M OSI	SA11_FS B	-	UART7_CTS	TIM14_C H1	QUADSP I_BK1_IO1	-	-	-	EVEN TOUT	
	PF10	-	-	-	-	-	-	-	-	-	-	-	-	DCMI_D 11	LCD_DE	EVEN TOUT	
	PF11	-	-	-	-	-	-	SP15_M OSI	-	-	-	-	FMC_SD NRAS	DCMI_D 12	-	EVEN TOUT	
	PF12	-	-	-	-	-	-	-	-	-	-	-	FMC_A6	-	-	EVEN TOUT	

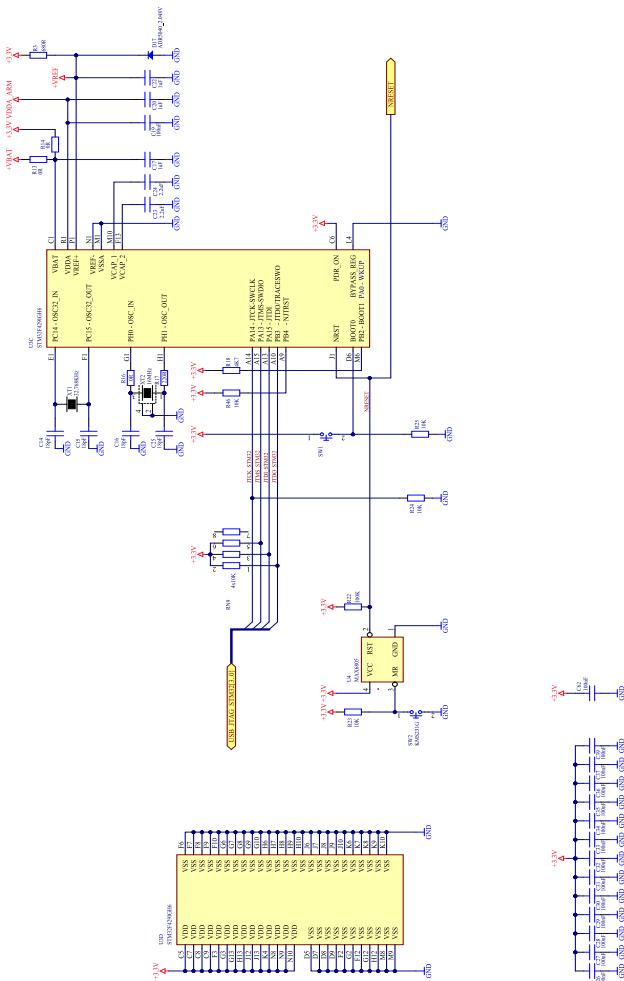
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15		
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SAI1	SPI2/3/U ART6/U SART1/2/ 3/UARTS/ SPDIFRX	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	CAN1/2/T IM12/3/ 14QUAD SPI/LCD	SAI2/QU ADSPPIO ADSPQD G2_HSI OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS		
Port F	PF13	-	-	-	-	I2C4_SM BA	-	-	-	-	-	-	-	FMC_A7	-	-	EVEN TOUT		
	PF14	-	-	-	-	I2C4_SC L	-	-	-	-	-	-	-	FMC_A8	-	-	EVEN TOUT		
	PF15	-	-	-	-	I2C4_SD A	-	-	-	-	-	-	-	FMC_A9	-	-	EVEN TOUT		
Port G	PG0	-	-	-	-	-	-	-	-	-	-	-	-	FMC_A1 0	-	-	EVEN TOUT		
	PG1	-	-	-	-	-	-	-	-	-	-	-	-	FMC_A1 1	-	-	EVEN TOUT		
	PG2	-	-	-	-	-	-	-	-	-	-	-	-	FMC_A1 2	-	-	EVEN TOUT		
	PG3	-	-	-	-	-	-	-	-	-	-	-	-	FMC_A1 3	-	-	EVEN TOUT		
	PG4	-	-	-	-	-	-	-	-	-	-	-	-	FMC_A1 4/FMC BA0	-	-	EVEN TOUT		
	PG5	-	-	-	-	-	-	-	-	-	-	-	-	FMC_A1 5/FMC BA1	-	-	EVEN TOUT		
	PG6	-	-	-	-	-	-	-	-	-	-	-	-	DCMI_D 12	LCD_R7	EVEN TOUT			
	PG7	-	-	-	-	-	-	-	USART6 _CK	-	-	-	-	FMC_IN T	DCMI_D 13	LCD_CL K	EVEN TOUT		
	PG8	-	-	-	-	SPI6_NS S	-	SPDIFRX _IN2	USART6 _RTS	-	-	-	-	ETH_PPS _OUT	FMC_SD CLK	-	EVEN TOUT		
	PG9	-	-	-	-	-	-	SPDIFRX _IN3	USART6 _RX	QUADSP L_BK2_IO 2	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	SPI2/FS B	SPI2/FS B	FMC_NE 2/FMC NCE	DCMI_V SYNC	-
Port H	PG10	-	-	-	-	-	-	-	-	-	LCD_G3	SPI2/SD B	-	FMC_NE 3	DCMI_D 2	LCD_B2	EVEN TOUT		
	PH0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT		
	PH1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT		
	PH2	-	-	-	LPTIM1_I N2	-	-	-	-	-	QUADSP L_BK2_IO 0	SPI2/SC K_B	ETH_MII CRS	FMC_SD CKE0	-	LCD_R0	EVEN TOUT		
	PH3	-	-	-	-	-	-	-	-	-	QUADSP L_BK2_IO 1	SPI2/SC K_B	ETH_MII COL	FMC_SD NE0	-	LCD_R1	EVEN TOUT		
	PH4	-	-	-	-	I2C2_SC L	-	-	-	-	OTG_HS UIP1_NX T	-	-	-	-	-	EVEN TOUT		
	PH5	-	-	-	-	I2C2_SD A	SPI5_NS S	-	-	-	-	-	-	FMC_SD NWE	-	-	EVEN TOUT		
	PH6	-	-	-	-	I2C2_SM BA	SPI5_SC K	-	-	-	TIM2_C H1	-	ETH_MII RXD2	FMC_SD NE1	DCMI_D 8	-	EVEN TOUT		
	PH7	-	-	-	-	I2C3_SC L	SPI5_MI SO	-	-	-	-	-	ETH_MII RXD3	FMC_SD CKE1	DCMI_D 9	-	EVEN TOUT		

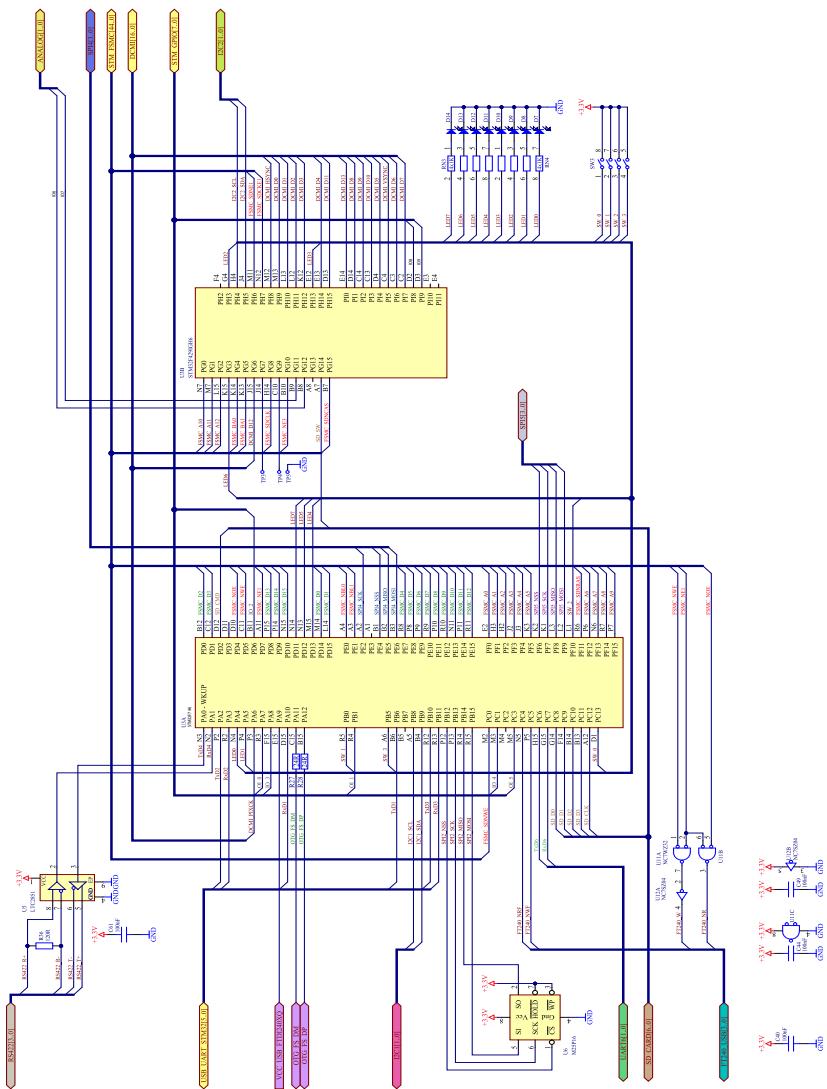
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15	
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SAI1	SPI2/3/U ART6/U SART1/2/ 3/UARTS/ SPDIFRX	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	CAN1/2/T IM12/3/ 14QUAD SPI/LCD	SPI2/3/U ART6/U RT4/5/7/8 /SPDIFR X	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS	
Port G	PG11	-	-	-	-	-	-	-	SPDIFRX _INO	-	-	-	-	ETH_MII TX_EN/E TH_RMI/ TX_EN	-	DCMI_D 3	LCD_B3	EVEN TOUT
	PG12	-	-	-	LPTIM1_I N1	-	SPI6_MI SO	-	SPDIFRX _IN1	USART6 _RTS	LCD_B4	-	-	FMC_NE 4	-	LCD_B1	EVEN TOUT	
	PG13	TRACE D0	-	-	LPTIM1_ OUT	-	SPI6_SC K	-	-	USART6 _CTS	-	-	-	ETH_MII TXD0/ET H_RMI/ T_XD0	-	LCD_R0	EVEN TOUT	
	PG14	TRACE D1	-	-	LPTIM1_E TR	-	SPI6_M OSI	-	-	USART6 _TX	QUADSP L_BK2_IO 3	-	-	ETH_MII TXD1/ET H_RMI/ T_XD1	-	LCD_B0	EVEN TOUT	
	PG15	-	-	-	-	-	-	-	-	USART6 _CTS	-	-	-	FMC_SD NCAS	DCMI_D 13	-	EVEN TOUT	
Port H	PH0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT	
	PH1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT	
	PH2	-	-	-	LPTIM1_I N2	-	-	-	-	-	QUADSP L_BK2_IO 0	SPI2/SC K_B	ETH_MII CRS	FMC_SD CKE0	-	LCD_R0	EVEN TOUT	
	PH3	-	-	-	-	-	-	-	-	-	QUADSP L_BK2_IO 1	SPI2/SC K_B	ETH_MII COL	FMC_SD NE0	-	LCD_R1	EVEN TOUT	
	PH4	-	-	-	-	I2C2_SC L	-	-	-	-	OTG_HS UIP1_NX T	-	-	-	-	-	EVEN TOUT	
	PH5	-	-	-	-	I2C2_SD A	SPI5_NS S	-	-	-	-	-	-	FMC_SD NWE	-	-	EVEN TOUT	
	PH6	-	-	-	-	I2C2_SM BA	SPI5_SC K	-	-	-	TIM2_C H1	-	ETH_MII RXD2	FMC_SD NE1	DCMI_D 8	-	EVEN TOUT	
	PH7	-	-	-	-	I2C3_SC L	SPI5_MI SO	-	-	-	-	-	ETH_MII RXD3	FMC_SD CKE1	DCMI_D 9	-	EVEN TOUT	

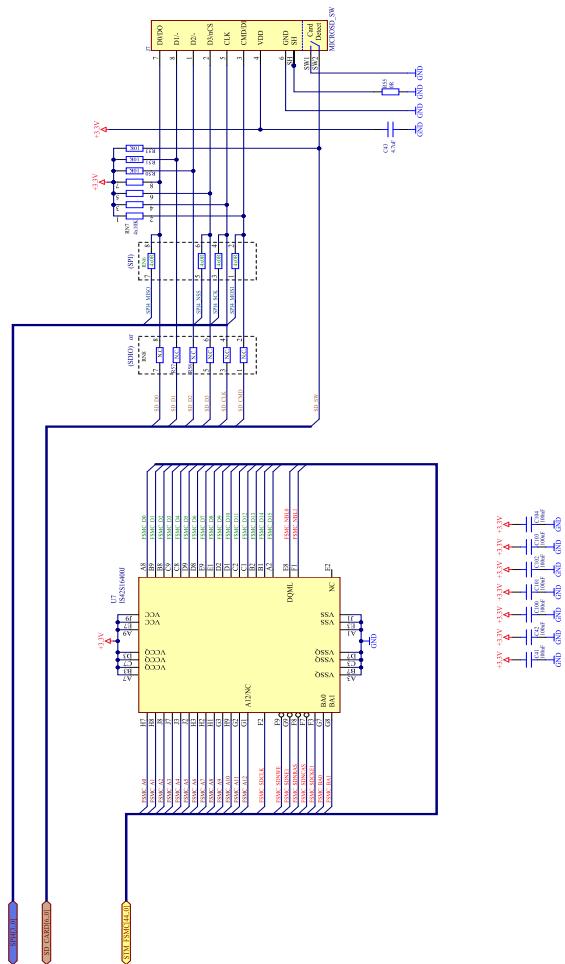
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SA1	SPI2/3/U SART1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/UA RT4/5/7/8 /SPDIFR X	CAN1/2/T IM12/13/ 14/QUAD SPI/LCD	SAI2/QU ADSPPIO TC2_H/S/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port H	PH8	-	-	-	-	I2C3_SD_A	-	-	-	-	-	-	FMC_D1_6	DCMI_H SYNC	LCD_R2	EVEN TOUT	
	PH9	-	-	-	-	I2C3_SM_BA	-	-	-	TIM12_C H2	-	-	FMC_D1_7	DCMI_D 0	LCD_R3	EVEN TOUT	
	PH10	-	-	TIM5_C H1	-	I2C4_SM_BA	-	-	-	-	-	-	FMC_D1_8	DCMI_D 1	LCD_R4	EVEN TOUT	
	PH11	-	-	TIM5_C H2	-	I2C4_SC_L	-	-	-	-	-	-	FMC_D1_9	DCMI_D 2	LCD_R5	EVEN TOUT	
	PH12	-	-	TIM5_C H3	-	I2C4_SD_A	-	-	-	-	-	-	FMC_D2_0	DCMI_D 3	LCD_R6	EVEN TOUT	
	PH13	-	-	-	TIM8_CH 1N	-	-	-	-	CAN1_T X	-	-	FMC_D2_1	-	LCD_G2	EVEN TOUT	
	PH14	-	-	-	TIM8_CH 2N	-	-	-	-	-	-	-	FMC_D2_2	DCMI_D 4	LCD_G3	EVEN TOUT	
	PH15	-	-	-	TIM8_CH 3N	-	-	-	-	-	-	-	FMC_D2_3	DCMI_D 11	LCD_G4	EVEN TOUT	
Port I	PI0	-	-	TIM5_C H4	-	-	SPI2_NS S/252_Ws	-	-	-	-	-	FMC_D2_4	DCMI_D 13	LCD_G5	EVEN TOUT	
	PI1	-	-	-	TIM8_BKI N2	-	SPI2_SC K/252_Ck	-	-	-	-	-	FMC_D2_5	DCMI_D 8	LCD_G6	EVEN TOUT	
	PI2	-	-	-	TIM8_CH 4	-	SPI2_MI_SO	-	-	-	-	-	FMC_D2_6	DCMI_D 9	LCD_G7	EVEN TOUT	
	PI3	-	-	-	TIM8_ET R	-	SPI2_M OSI/2/S2_SD	-	-	-	-	-	FMC_D2_7	DCMI_D 10	-	EVEN TOUT	
	PI4	-	-	-	TIM8_BKI N	-	-	-	-	-	SAI2_MC K_A	-	FMC_NB L2	DCMI_D 5	LCD_B4	EVEN TOUT	
	PI5	-	-	-	TIM8_CH 1	-	-	-	-	-	SAI2_SC K_A	-	FMC_NB L3	DCMI_V SYNC	LCD_B5	EVEN TOUT	
	PI6	-	-	-	TIM8_CH 2	-	-	-	-	-	SAI2_SD_ A	-	FMC_D2_8	DCMI_D 6	LCD_B6	EVEN TOUT	

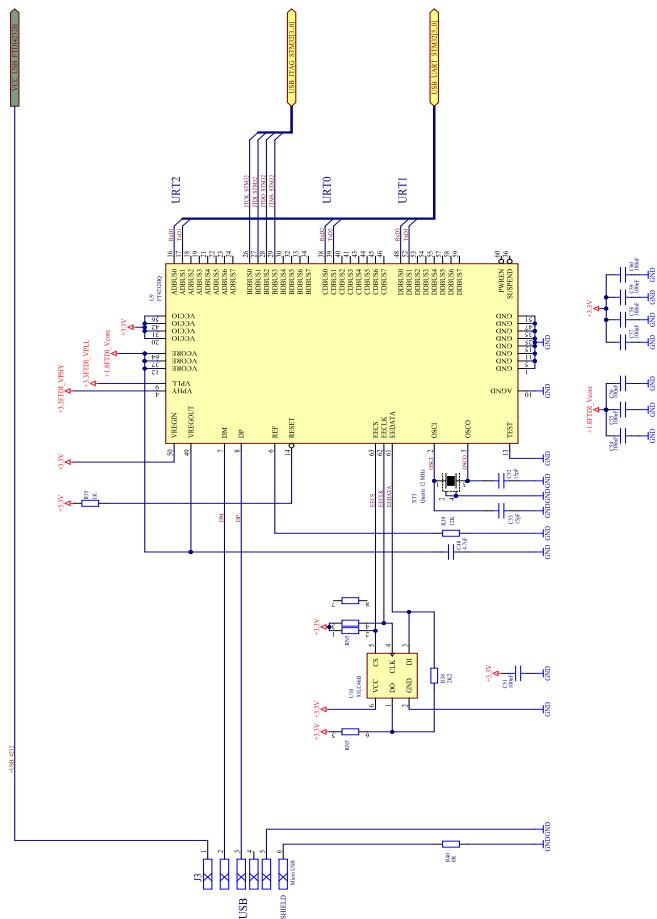
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/ 11/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SA1	SPI2/3/U SART1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/UA RT4/5/7/8 /SPDIFR X	CAN1/2/T IM12/13/ 14/QUAD SPI/LCD	SAI2/QU ADSPPIO TC2_H/S/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS
Port I	PI7	-	-	-	TIM8_CH 3	-	-	-	-	-	SAI2_FS_A	-	FMC_D2_9	DCMI_D 7	LCD_B7	EVEN TOUT	
	PI8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVEN TOUT	
	PI9	-	-	-	-	-	-	-	-	CAN1_R X	-	-	FMC_D3_0	-	LCD_VS YNC	EVEN TOUT	
	PI10	-	-	-	-	-	-	-	-	-	-	-	ETH_MII_RX_ER	FMC_D3_1	-	LCD_HS YNC	EVEN TOUT
	PI11	-	-	-	-	-	-	-	-	-	OTG_HS_ULPI_DIR	-	-	-	-	EVEN TOUT	
	PI12	-	-	-	-	-	-	-	-	-	-	-	-	-	LCD_HS YNC	EVEN TOUT	
	PI13	-	-	-	-	-	-	-	-	-	-	-	-	-	LCD_VS YNC	EVEN TOUT	
	PI14	-	-	-	-	-	-	-	-	-	-	-	-	-	LCD_CL K	EVEN TOUT	
	PI15	-	-	-	-	-	-	-	-	-	-	-	-	-	LCD_R0	EVEN TOUT	

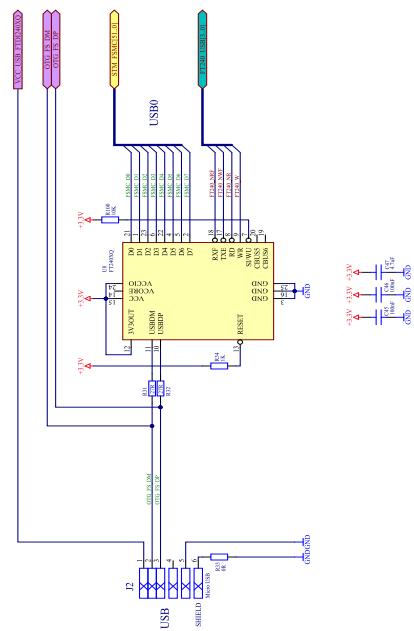


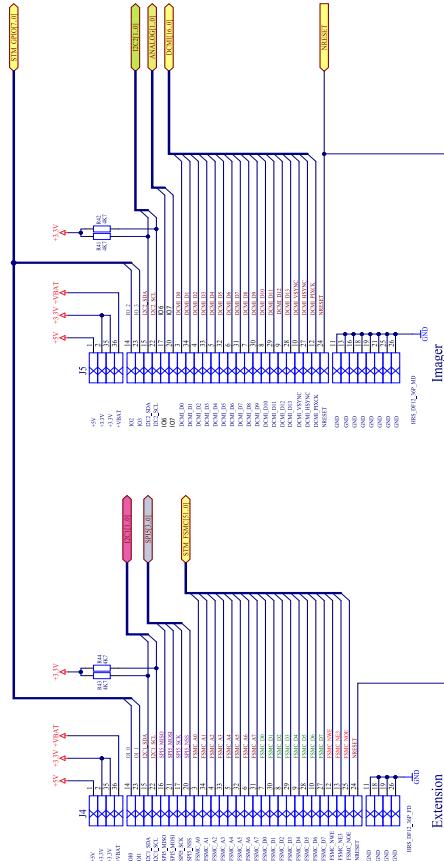




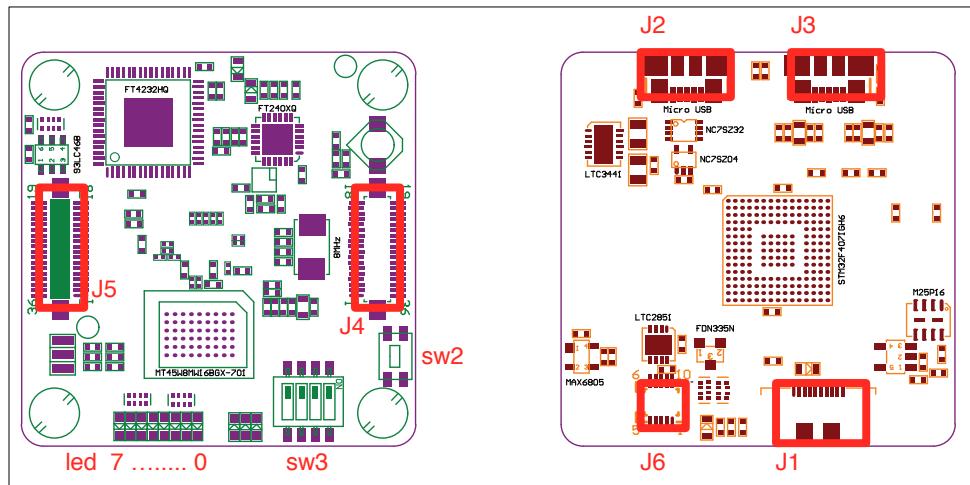






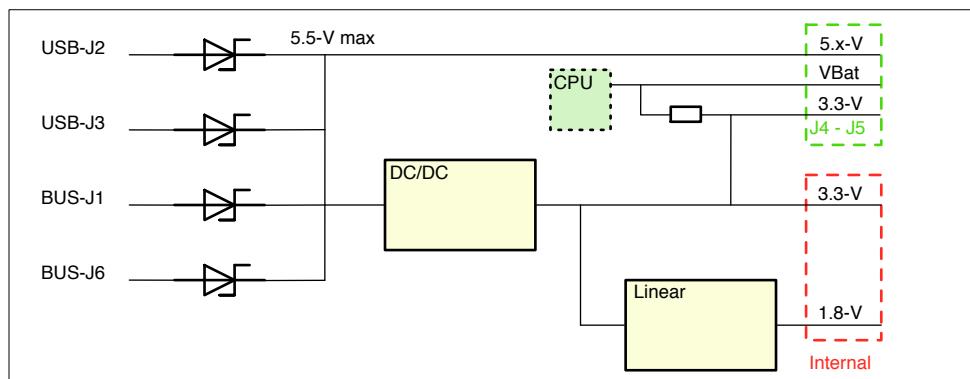


7.4. PCB and the interfaces



7.4.1. The Power supply

The board can be powered with voltages in the range of 4.2...5.5-V. Two step-downs are available to generate on the board the necessary voltage of 3.3-V and 1.8-V. The main power supply can be delivered by the **USB** connectors (**J2** and **J3**) as well as by the main industrial interface (**J1**) and by **J6**.



7.4.2. JTAG and the (FTDI) communications

The JTAG connection is ensured by the **USB/FTDI** connected via **J3**. The JTAG is used for programming the internal Flash (by **openocd**) and for using the **gdb**.

Three **uarts** are available by the **USB/FTDI** connected via **J3**. The **uart1**, **uart2** and **uart3** are logically connected to the communication channels **KURT2**, **KURT0** and **KURT1**.

An high-speed **FIFO** port is available by the **USB/FTDI** connected via **J2**. This USB channel can be replaced by the one included in the μController.

	Function	Connector	Cortex-M4F / μKernel		
FTDI chip			Physical	Logical	μKernel
FT-4232 Channel a	UART	J3 micro usb	UART1	KURT2	Console
FT-4232 Channel b	JTAG	J3 micro usb	JTAG	-	-
FT-4232 Channel c	UART	J3 micro usb	UART2	KURT0	Console (default)
FT-4232 Channel d	UART	J3 micro usb	UART3	KURT1	Console
FT-240	Parallel	J2 micro usb	0x60000000		Images / Tracing

7.4.3. The main industrial interface

An RS422 **uart (uart4)** or an **spi (spi5)** is available on the connector **J1**. The selection between these two configurations is done by **RN1** and **RN2**.

	Function	Connector	Cortex-M4F / μKernel		
			Physical	Logical	μKernel
Power supply	5.0-V	J1 - 1, 2, 3	-	-	-
Power supply	GND	J1 - 8, 9, 10	-	-	-
RS422 RX +	UART	J1 - 4	UART4	KURT3	Console
RS422 RX -	UART	J1 - 5	UART4	KURT3	Console
RS422 TX +	UART	J1 - 7	UART4	KURT3	Console
RS422 TX -	UART	J1 - 6	UART4	KURT3	Console
SPI - MISO	SPI	J1 - 4	SPI5	-	-
SPI - MOSI	SPI	J1 - 5	SPI5	-	-
SPI - CLK	SPI	J1 - 6	SPI5	-	-
/SPI - CS	SPI	J1 - 7	SPI5	-	-

7.4.4. The accumulator and the Bluetooth interface

An **uart (uart6)** is available on the connector **J6**. Four **GPIO (IO4, IO8, IO9 and OI5)** are also available.

	Function	Connector	Cortex-M4F / μKernel		
			Physical	Logical	μKernel
Power supply	5.0-V	J6 - 1, 2	-	-	-
Power supply	GND	J6 - 3, 4	-	-	-
UART TX	UART	J6 - 8	UART6	KURT4	Console
UART RX	UART	J6 - 9	UART6	KURT4	Console
I/O (input)	IO4, IO8, IO9	J6 - 10, 7, 6	I/O	-	-
I/O (output)	OI5	J6 - 5	I/O	-	-

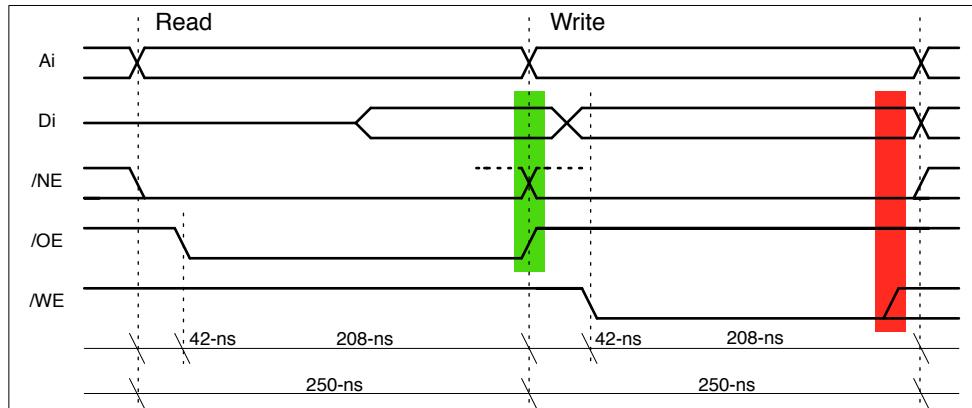
7.4.5. The extension bus

This bus (**J4** and **J5**) includes many functionalities. Here is the detail:

1. 8-bit wide CPU bus (256 addresses).
2. One SPI (spi5) and two I2C (i2C1 and i2C2).
3. Six GPIO (O10, O11, IO2, IO3, O16, O17).
4. A full DCMI bus (imager bus).

	Function	Connector	Cortex-M7F / μKernel		
			Physical	Logical	μKernel
Power supply	5.0-V	J5 - 1	-	-	-
Power supply	3.3-V	J5 - 2, 35	-	-	-
Power supply	VBat	J5 - 36	-	-	-
Power supply	GND	J5 - 11, 13, 16, 18, 19, 21, 25, 26	-	-	-
I/O (input)	IO2	J5 - 14	I/O	-	-
I/O (input)	IO3	J5 - 23	I/O	-	-
I2C - SDA	I2C	J5 - 15	I2C2	-	i2c0
I2C - SCL	I2C	J5 - 22	I2C2	-	i2c0
I/O (Output)	O16	J5 - 17	I/O	-	-
I/O (Output)	O17	J5 - 20	I/O	-	-
DCMI0 .. DCMI13	Imager	J5 - 3, 34, 4, 33, 5, 32, 6, 31, 7, 30, 8, 29, 9, 28	DCMI	-	img0
VSYNC	Imager	J5 - 10	DCMI	-	img0
H SYNC	Imager	J5 - 27	DCMI	-	img0
PIXCLK	Imager	J5 - 12	DCMI	-	img0
/RESET	CPU	J5 - 24	CPU	-	-

	Function	Connector	Cortex-M4F / μKernel		
			Physical	Logical	μKernel
Power supply	5.0-V	J4 - 1	-	-	-
Power supply	3.3-V	J4 - 2, 35	-	-	-
Power supply	VBat	J4 - 36	-	-	-
Power supply	GND	J4 - 11, 18, 19, 26	-	-	-
I/O (output)	OI0	J4 - 14	I/O	-	-
I/O (output)	OI1	J4 - 23	I/O	-	-
I2C - SDA	I2C	J4 - 15	I2C1	-	i2c1
I2C - SCL	I2C	J4 - 22	I2C1	-	i2c1
SPI - MISO	SPI	J4 - 16	SPI5	-	-
SPI - MOSI	SPI	J4 - 21	SPI5	-	-
SPI - SCK	SPI	J4 - 17	SPI5	-	-
SPI - NSS	SPI	J4 - 20	SPI5	-	-
A0 .. A7	CPU	J4 - 3, 34, 4, 33, 5, 32, 6, 31	FMC	-	misc
D0 .. D7	CPU	J4 - 7, 30, 8, 29, 9, 28, 10, 27	FMC	-	misc
/WE	CPU	J4 - 12	FMC	-	misc
/NE	CPU	J4 - 13	FMC	-	misc
/OE	CPU	J4 - 25	FMC	-	misc
/RESET	CPU	J5 - 24	FMC	-	misc



This timing is set by default by the `init.c` module. If for any reasons it is necessary to modify it, here is an example of routine for this purpose.

```
/*
 * \brief _FMC_Configuration
 *
 * - FMC configuration
 */
static void _FMC_Configuration(void) {
    RCC->AHB3ENR |= RCC_AHB3ENR_FSMCEN; // Turn on the FMCEN

    // FSMC bank 1 & CE3 configuration in the asynchronous mode
    // - Address/Data non multiplexed
    // - EXTENSION (200-ns), 8 bits
    // - Write operations are enabled for the bank by the FMC (default after reset)
    // - Write operations are always performed in asynchronous mode

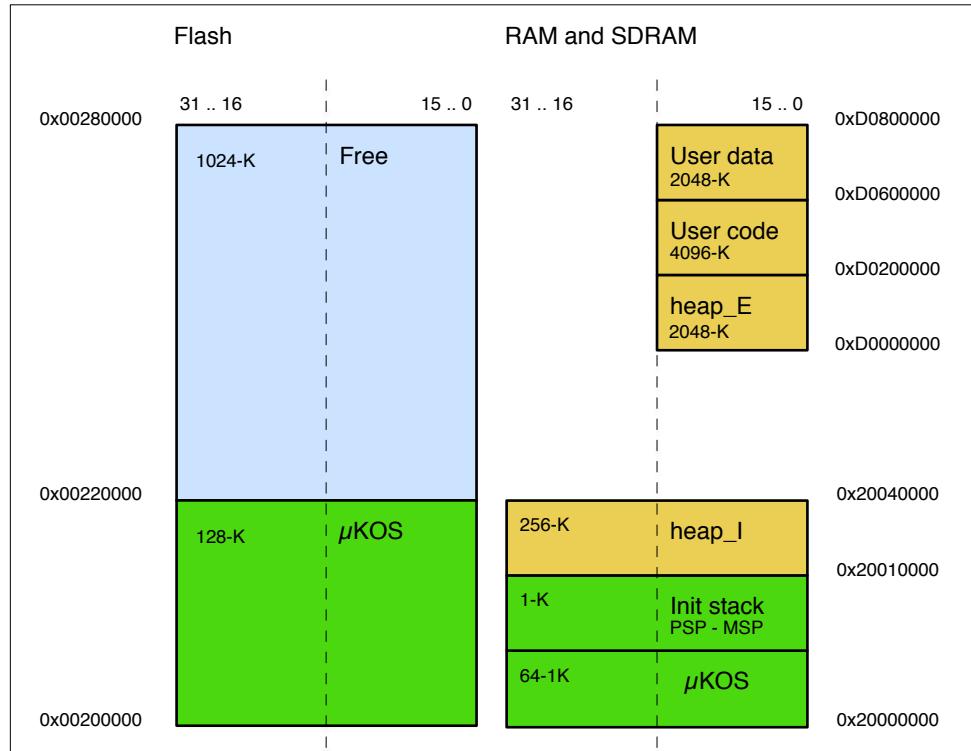
    FMC_Bank1->BCR1 = FMC_BCR_WREN; // Enable the write enable

    FMC_Bank1->BTR1 = (0<<28) // Access mode A
                           | (0<<24) // DATLAT (do not care)
                           | (0<<20) // CLKDIV (do not care)
                           | (0<<16) // BUSTURN of 0 clks
                           | (35<<8) // DATAST of 35 clks
                           | (0<<4) // ADDHLD (do not care)
                           | (7<<0); // ADDSET 7 of clks (Address setup)

    FMC_Bank1->BCR1 |= FMC_BCR_MBKEN; // Memory bank enable
}
```

7.5. Memory mapping and CPU initialization

7.5.1. μKOS mapping



7.5.2. The CPU initialization

```
/*
; init.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:     Franzi Edo.  The 2014-02-25      Add 5.95ns to the SDRAM timing
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 2                $: Revision of last commit
; $Date::: 2016-02-16 21:34:30#$: Date of last commit
;
; Project:    uKOS
; Goal:       Low level init for the uKOS Baphomet_746 module.
;
;           !!! This code HAS not to contain static data.
;           !!! It is called before to copy and to initialize
;           !!! the variable into the RAM.
;
;           (c) 1992-2017, Franzi Edo.
; -----
;
; Franzi Edo.
; 5-Route de Cheseaux
; CH 1400 Cheseaux-Noréaz
; edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
; -----
*/

```

```
#include      <uKOS.h>

#define      __CACHE_I__          // With the instruction cache
#define      __CACHE_D__          // With the data cache

extern uint32_t      _stEXRAM, _lnEXRAM;

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) = \
    "init           First hardware initializations.          (c) EFr-2017";
LOC_CONST_STRG(aStrHelp[])         = \
    "Init\n"
"====\n\n"

"This code places in a quite state the hardware resources.\n\n";
```

```

// GPIO config macro
// =====

#define      CNFGPIO(port, \
    a15, a14, a13, a12, a11, a10, a9, a8, a7, a6, a5, a4, a3, a2, a1, a0, \
    b15, b14, b13, b12, b11, b10, b9, b8, b7, b6, b5, b4, b3, b2, b1, b0, \
    c15, c14, c13, c12, c11, c10, c9, c8, c7, c6, c5, c4, c3, c2, c1, c0, \
    d15, d14, d13, d12, d11, d10, d9, d8, d7, d6, d5, d4, d3, d2, d1, d0, \
    e15, e14, e13, e12, e11, e10, e9, e8, e7, e6, e5, e4, e3, e2, e1, e0, \
    f15, f14, f13, f12, f11, f10, f9, f8, f7, f6, f5, f4, f3, f2, f1, f0) \
GPIO##port->AFR[1] = (d15<<28) | (d14<<24) | (d13<<20) | (d12<<16) | \
(d11<<12) | (d10<<8) | (d9<<4) | (d8<<0); \
GPIO##port->AFR[0] = (d7<<28) | (d6<<24) | (d5<<20) | (d4<<16) | \
(d3<<12) | (d2<<8) | (d1<<4) | (d0<<0); \
GPIO##port->OSPEEDR = (b15<<30) | (b14<<28) | (b13<<26) | (b12<<24) | \
(b11<<22) | (b10<<20) | (b9<<18) | (b8<<16) | \
(b7<<14) | (b6<<12) | (b5<<10) | (b4<<8) | \
(b3<<6) | (b2<<4) | (b1<<2) | (b0<<0); \
GPIO##port->OTYPER = (e15<<15) | (e14<<14) | (e13<<13) | (e12<<12) | \
(e11<<11) | (e10<<10) | (e9<<9) | (e8<<8) | \
(e7<<7) | (e6<<6) | (e5<<5) | (e4<<4) | \
(e3<<3) | (e2<<2) | (e1<<1) | (e0<<0); \
GPIO##port->MODER = (a15<<30) | (a14<<28) | (a13<<26) | (a12<<24) | \
(a11<<22) | (a10<<20) | (a9<<18) | (a8<<16) | \
(a7<<14) | (a6<<12) | (a5<<10) | (a4<<8) | \
(a3<<6) | (a2<<4) | (a1<<2) | (a0<<0); \
GPIO##port->PUPDR = (c15<<30) | (c14<<28) | (c13<<26) | (c12<<24) | \
(c11<<22) | (c10<<20) | (c9<<18) | (c8<<16) | \
(c7<<14) | (c6<<12) | (c5<<10) | (c4<<8) | \
(c3<<6) | (c2<<4) | (c1<<2) | (c0<<0); \
GPIO##port->ODR = (f15<<15) | (f14<<14) | (f13<<13) | (f12<<12) | \
(f11<<11) | (f10<<10) | (f9<<9) | (f8<<8) | \
(f7<<7) | (f6<<6) | (f5<<5) | (f4<<4) | \
(f3<<3) | (f2<<2) | (f1<<1) | (f0<<0);

// Prototypes
// =====

static void _RCC_Configuration(void);
static void _FPU_Configuration(void);
static void _GPIO_Configuration(void);
static void _FMC_Configuration(void);
static void _CACHE_Enable(void);

// Module specifications (See the modules.h)
// =====

MODULE(Init, KIDSTARTUP, KINITNUM, 0, "1.0", (1<<BSHOW));

```

```
/*
 * \brief init_init
 *
 * - Initialize some basic periphs
 * - GPIO, watchdog, SDRAM
 *
 * \param[in] -
 * \param[out] -
 *
 */
void    init_init(void) {

    _RCC_Configuration();                                // Clock Config
    _FPU_Configuration();                               // FPU Enabled
    _GPIO_Configuration();                            // GPIO Config
    _FMC_Configuration();                           // FMC Config
    _CACHE_Enable();                                 // Enable the instruction/data cache
}

/*
 * \brief _FPU_Configuration
 *
 * - Enable the FPU
 *
 */
static void    _FPU_Configuration(void) {

    SCB->CPACR |= ((SCB_CPACR_FPAF<<20) | (SCB_CPACR_FPAF<<22));
    FPU->FPCCR |= ((1<<FPU_FPCCR_AS PEN) | (1<<FPU_FPCCR_LSPEN));
}

/*
 * \brief _GPIO_Configuration
 *
 * - GPIO configuration
 *
 */
static void    _GPIO_Configuration(void) {

    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;           // Turn on the GPIOA
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;          // Turn on the GPIOB
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;          // Turn on the GPIOC
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;          // Turn on the GPIOD
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOEEN;          // Turn on the GPIOE
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOFEN;          // Turn on the GPIOF
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOGEN;          // Turn on the GPIOG
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOHEN;          // Turn on the GPIOH
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOIEN;          // Turn on the GPIOI

// Init all the GPIO A, B, C, D, E, F, G, H, I
}
```

```

// PA00, AL, 50-MHz, Push-pull    USART4_TX      AF8
// PA01, AL, 50-MHz, Pull-up      USART4_RX      AF8
// PA02, AL, 50-MHz, Push-pull    USART2_TX      AF7
// PA03, AL, 50-MHz, Pull-up      USART2_RX      AF7
// PA04, OU, 50-MHz, Push-pull    LED0          AF15
// PA05, OU, 50-MHz, Push-pull    LED5          AF15
// PA06, AL, 50-MHz, Push-pull    DCMI_PIXCK   AF13
// PA07, OU, 50-MHz, Push-pull    OIO           AF15
// PA08, IN, 50-MHz, Pull-up     IO3           AF15
// PA09, IN, 50-MHz, Pull-up     VBUS_FS!!    AF15
// PA10, AL, 50-MHz, Pull-up     USART1_RX      AF7
// PA11, IN, 50-MHz, Pull-up     OTG_DM!!    AF15
// PA12, IN, 50-MHz, Pull-up     OTG_DP!!    AF15
// PA13, AL, 50-MHz, Pull-up     TMS          AF0
// PA14, AL, 50-MHz, Pull-down   TCK          AF0
// PA15, AL, 50-MHz, Pull-up     TDI          AF0

CNFGPIO(A,KAL,KAL,KAL,KIN,KIN,KIN,KIN,KOU,KAL,KOU,KAL,KAL,KAL,KAL, \
         K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50, \
         KPU,KPD,KPU,KPU,KPU,KPU,KPU,KPU,KPU,KNO,KNO,KNO,KNO,KPU,KNO,KPU,KNO, \
         A00,A00,A00,A15,A15,A07,A15,A15,A13,A15,A15,A07,A07,A08,A08, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

// PB00, IN, 50-MHz, Pull-down   SW1          AF15
// PB01, OU, 50-MHz, Push-pull   OI1          AF15
// PB02, IN, 50-MHz, Pull-up     Boot1        AF15
// PB03, AL, 50-MHz, Push-pull   TDO          AF0
// PB04, IN, 50-MHz, Pull-up     NJTRST      AF0
// PB05, IN, 50-MHz, Pull-down   SW3          AF15
// PB06, AL, 50-MHz, Push-pull   USART1_TX    AF7
// PB07, IN, 50-MHz, Pull-up     -            AF15
// PB08, AL, 50-MHz, Open DU    I2C1_SCL    AF4
// PB09, AL, 50-MHz, Open DU    I2C1_SDA    AF4
// PB10, AL, 50-MHz, Push-pull   USART3_TX    AF7
// PB11, AL, 50-MHz, Pull-up     USART3_RX    AF7
// PB12, OU, 50-MHz, Push-pull   SPI2_NSS    AF15  BSELEEPROM
// PB13, AL, 50-MHz, Push-pull   SPI2_SCK    AF5
// PB14, AL, 50-MHz, Pull-up     SPI2_MISO   AF5
// PB15, AL, 50-MHz, Push-pull   SPI2_MOSI   AF5

CNFGPIO(B,KAL,KAL,KAL,KOU,KAL,KAL,KAL,KAL,KIN,KIN,KIN,KAL,KIN,KIN,KOU,KIN, \
         K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50, \
         KNO,KPU,KNO,KNO,KPU,KNO,KPU,KPU,KPU,KNO,KPD,KPU,KNO,KPU,KNO,KPD, \
         A05,A05,A05,A15,A07,A07,A04,A04,A15,A07,A15,A00,A00,A15,A15,A15, \
         KPP,KPP,KPP,KPP,KPP,KOD,KOD,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

```

```

// PC00, AL, 99-MHz, Push-pull    SDNWE      AF12
// PC01, IN, 50-MHz, Pull-up      IO4        AF15
// PC02, IN, 50-MHz, Pull-up      -          AF15
// PC03, OU, 50-MHz, Push-pull    OI5        AF15
// PC04, IN, 50-MHz, Pull-up      FTDI_nRF    AF15    BIRQUSBR
// PC05, IN, 50-MHz, Pull-up      FTDI_nWR    AF15    BIRQUSBW
// PC06, AL, 50-MHz, Push-pull    USART6_TX   AF8
// PC07, AL, 50-MHz, Pull-up      USART6_RX   AF8
// PC08, AL, 50-MHz, Pull-up      SDIO_D0     AF12
// PC09, AL, 50-MHz, Pull-up      SDIO_D1     AF12
// PC10, AL, 50-MHz, Pull-up      SDIO_D2     AF12
// PC11, AL, 50-MHz, Pull-up      SDIO_D3     AF12
// PC12, AL, 50-MHz, Push-pull   SDIO_CLK    AF12
// PC13, IN, 50-MHz, Pull-down   SW0        AF15
// PC14, AL, 50-MHz, -           OSC        AF0
// PC15, AL, 50-MHz, -           OSC        AF0

CNFGPIO(C,KAL,KAL,KIN,KAL,KAL,KAL,KAL,KAL,KAL,KIN,KIN,KIN,KIN,KAL, \
         K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K99, \
         KNO,KNO,KPD,KNO,KPU,KPU,KPU,KPU,KPU,KNO,KPU,KPU,KNO,KPU,KPU,KNO, \
         A00,A00,A15,A12,A12,A12,A12,A12,A08,A08,A15,A15,A15,A15,A15,A12, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

// PD00, AL, 99-MHz, Push-pull   FMC_D2      AF12
// PD01, AL, 99-MHz, Push-pull   FMC_D3      AF12
// PD02, AL, 50-MHz, Push-pull   SDIO_CMD    AF12
// PD03, IN, 50-MHz, Push-pull   -          AF15
// PD04, AL, 99-MHz, Push-pull   FMC_NOE    AF12
// PD05, AL, 99-MHz, Push-pull   FMC_NWE    AF12
// PD06, IN, 50-MHz, Pull-up    IO2        AF15
// PD07, AL, 99-MHz, Push-pull   FMC_NE1    AF12
// PD08, AL, 99-MHz, Push-pull   FMC_D13    AF12
// PD09, AL, 99-MHz, Push-pull   FMC_D14    AF12
// PD10, AL, 99-MHz, Push-pull   FMC_D15    AF12
// PD11, OU, 50-MHz, Push-pull   LED7       AF15
// PD12, OU, 50-MHz, Push-pull   LED5       AF15
// PD13, OU, 50-MHz, Push-pull   LED4       AF15
// PD14, AL, 99-MHz, Push-pull   FMC_D0     AF12
// PD15, AL, 99-MHz, Push-pull   FMC_D1     AF12

CNFGPIO(D,KAL,KAL,KOU,KOU,KOU,KAL,KAL,KAL,KAL,KAL,KIN,KAL,KAL,KIN,KAL,KAL, \
         K99,K99,K50,K50,K99,K99,K99,K99,K50,K99,K99,K99,K50,K99,K99, \
         KNO,KNO,KNO,KNO,KNO,KNO,KNO,KPU,KNO,KPU,KNO,KNO,KNO, \
         A12,A12,A15,A15,A15,A12,A12,A12,A12,A15,A12,A12,A15,A12,A12,A12, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

```

```

// PE00, AL, 99-MHz, Push-pull      FMC_NBL0      AF12
// PE01, AL, 99-MHz, Push-pull      FMC_NBL1      AF12
// PE02, AL, 50-MHz, Push-pull      SPI4_SCK      AF5
// PE03, IN, 50-MHz, Pull-up        -             AF12
// PE04, OU, 50-MHz, Push-pull      SPI4 NSS      AF15      BSELSDCARD
// PE05, AL, 50-MHz, Pull-up        SPI4_MISO     AF5
// PE06, AL, 50-MHz, Push-pull      SPI4 MOSI     AF5
// PE07, AL, 99-MHz, Push-pull      FMC_D4       AF12
// PE08, AL, 99-MHz, Push-pull      FMC_D5       AF12
// PE09, AL, 99-MHz, Push-pull      FMC_D6       AF12
// PE10, AL, 99-MHz, Push-pull      FMC_D7       AF12
// PE11, AL, 99-MHz, Push-pull      FMC_D8       AF12
// PE12, AL, 99-MHz, Push-pull      FMC_D9       AF12
// PE13, AL, 99-MHz, Push-pull      FMC_D10      AF12
// PE14, AL, 99-MHz, Push-pull      FMC_D11      AF12
// PE15, AL, 99-MHz, Push-pull      FMC_D12      AF12

CNFGPIO(E,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KOU,KIN,KAL,KAL,KAL, \
         K99,K99,K99,K99,K99,K99,K99,K99,K99,K99,K50,K50,K50,K50,K99,K99, \
         KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KPU,KNO,KPU,KNO,KPU,KPU, \
         A12,A12,A12,A12,A12,A12,A12,A12,A05,A05,A15,A15,A05,A12,A12, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0);

// PF00, AL, 99-MHz, Push-pull      FMC_A0       AF12
// PF01, AL, 99-MHz, Push-pull      FMC_A1       AF12
// PF02, AL, 99-MHz, Push-pull      FMC_A2       AF12
// PF03, AL, 99-MHz, Push-pull      FMC_A3       AF12
// PF04, AL, 99-MHz, Push-pull      FMC_A4       AF12
// PF05, AL, 99-MHz, Push-pull      FMC_A5       AF12
// PF06, OU, 50-MHz, Push-pull      SPI5 NSS     AF15      BSELADT7320
// PF07, AL, 50-MHz, Push-pull      SPI5 SCK     AF5
// PF08, AL, 50-MHz, Pull-up       SPI5 MISO     AF5
// PF09, AL, 50-MHz, Push-pull      SPI5 MOSI     AF5
// PF10, IN, 50-MHz, Pull-down     SW2          AF15
// PF11, AL, 99-MHz, Push-pull      SDNRAS      AF12
// PF12, AL, 99-MHz, Push-pull      FMC_A6       AF12
// PF13, AL, 99-MHz, Push-pull      FMC_A7       AF12
// PF14, AL, 99-MHz, Push-pull      FMC_A8       AF12
// PF15, AL, 99-MHz, Push-pull      FMC_A9       AF12

CNFGPIO(F,KAL,KAL,KAL,KAL,KAL,KIN,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL, \
         K99,K99,K99,K99,K99,K50,K50,K50,K50,K99,K99,K99,K99,K99,K99, \
         KNO,KNO,KNO,KNO,KNO,KPD,KNO,KPU,KNO,KNO,KNO,KNO,KNO,KNO,KNO, \
         A12,A12,A12,A12,A12,A15,A05,A05,A15,A12,A12,A12,A12,A12, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0);

```

```

// PG00, AL, 99-MHz, Push-pull      FMC_A10      AF12
// PG01, AL, 99-MHz, Push-pull      FMC_A11      AF12
// PG02, AL, 99-MHz, Push-pull      FMC_A12      AF12
// PG03, OU, 50-MHz, Push-pull      LED6        AF15
// PG04, AL, 99-MHz, Push-pull      BA0         AF12
// PG05, AL, 99-MHz, Push-pull      BA1         AF12
// PG06, AL, 50-MHz, Push-pull      DCMI_D12    AF13
// PG07, OU, 50-MHz, Push-pull      TP3         AF15
// PG08, AL, 99-MHz, Push-pull      SDCLK       AF12
// PG09, OU, 50-MHz, Push-pull      TP4         AF15
// PG10, AL, 99-MHz, Push-pull      FMC_NE3     AF12
// PG11, OU, 50-MHz, Push-pull      OI6         AF15
// PG12, OU, 50-MHz, Push-pull      OI7         AF15
// PG13, IN, 50-MHz, Pull-up       -           AF15
// PG14, IN, 50-MHz, Pull-up       CD          AF15      BNOTCARD
// PG15, AL, 99-MHz, Push-pull      SDNCAS     AF12

CNFGPIO(G,KAL,KIN,KIN,KOU,KAL,KAL,KOU,KAL,KAL,KAL,KAL,KOU,KAL,KAL,KAL, \
         K99,K50,K50,K50,K99,K50,K99,K50,K99,K50,K99,K50,K99,K99,K99, \
         KNO,KPU,KPU,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO, \
         A12,A15,A15,A15,A15,A12,A15,A12,A15,A13,A12,A12,A15,A12,A12,A12, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

// PH00, AL, 50-MHz, -              OSC          AF0
// PH01, AL, 50-MHz, -              OSC          AF0
// PH02, IN, 50-MHz, Pull-up       -           AF15
// PH03, OU, 50-MHz, Push-pull     LED2        AF15
// PH04, AL, 50-MHz, Open DU      I2C2_SCL   AF4
// PH05, AL, 50-MHz, Open DU      I2C2_SDA   AF4
// PH06, AL, 99-MHz, Push-pull     SDNEE1     AF12
// PH07, AL, 99-MHz, Push-pull     SDCKE1    AF12
// PH08, AL, 50-MHz, Push-pull     DCMI_HSYNC AF13
// PH09, AL, 50-MHz, Push-pull     DCMI_D0    AF13
// PH10, AL, 50-MHz, Push-pull     DCMI_D1    AF13
// PH11, AL, 50-MHz, Push-pull     DCMI_D2    AF13
// PH12, AL, 50-MHz, Push-pull     DCMI_D3    AF13
// PH13, OU, 50-MHz, Push-pull     LED3        AF15
// PH14, AL, 50-MHz, Push-pull     DCMI_D4    AF13
// PH15, AL, 50-MHz, Push-pull     DCMI_D11   AF13

CNFGPIO(H,KAL,KAL,KOU,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL, \
         K50,K50,K50,K50,K50,K50,K50,K50,K99,K99,K50,K50,K50,K50,K50, \
         KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KPU,KPU,KNO,KPU,KNO,KNO, \
         A13,A13,A15,A13,A13,A13,A13,A13,A12,A12,A04,A04,A15,A15,A00,A00, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KOD,KOD,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0);

```

```

// PIO0, AL, 50-MHz, Push-pull    DCMI_D13      AF13
// PIO1, AL, 50-MHz, Push-pull    DCMI_D8       AF13
// PIO2, AL, 50-MHz, Push-pull    DCMI_D9       AF13
// PIO3, AL, 50-MHz, Push-pull    DCMI_D10      AF13
// PIO4, AL, 50-MHz, Push-pull    DCMI_D5       AF13
// PIO5, AL, 50-MHz, Push-pull    DCMI_VSYNC    AF13
// PIO6, AL, 50-MHz, Push-pull    DCMI_D6       AF13
// PIO7, AL, 50-MHz, Push-pull    DCMI_D7       AF13
// PIO8, IN, 50-MHz, Pull-up     OI8          AF15
// PIO9, IN, 50-MHz, Pull-up     OI9          AF15
// PIO10, IN, 50-MHz, Pull-up    -           AF15
// PIO11, IN, 50-MHz, Pull-up    -           AF15
// PIO12, IN, 50-MHz, Pull-up    -           AF15
// PIO13, IN, 50-MHz, Pull-up    -           AF15
// PIO14, IN, 50-MHz, Pull-up    -           AF15
// PIO15, IN, 50-MHz, Pull-up    -           AF15

CNFGPIO(I,KIN,KIN,KIN,KIN,KIN,KIN,KOU,KAL,KAL,KAL,KAL,KAL,KAL,KAL,KAL, \
         K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50,K50, \
         KPU,KPU,KPU,KPU,KPU,KPU,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO,KNO, \
         A15,A15,A15,A15,A15,A15,A15,A13,A13,A13,A13,A13,A13,A13, \
         KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP,KPP, \
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

}

/*
 * \brief _RCC_Configuration
 *
 * - Clock, PLL configuration
 *   - The system Clock is configured as follow :
 *     - System Clock source      = PLL (HSE)
 *     - SYSCLK(Hz)              = 216000000
 *     - HCLK(Hz)                = 216000000
 *     - AHB Prescaler           = 1
 *     - APB1 Prescaler          = 8
 *     - APB2 Prescaler          = 4
 *     - HSE Frequency(Hz)       = 16000000
 *     - PLL_M                   = 16
 *     - PLL_N                   = 432
 *     - PLL_P                   = 2
 *     - PLL_Q                   = 9
 *     - VDD(V)                  = 3.3
 *     - Main regulator output voltage = Scale1 mode
 *     - Flash Latency(WS)        = 7
 *
 */
static void _RCC_Configuration(void) {
    RCC->CR      |= 0x00010001;                      // HSI & HSE on
    RCC->CFGR    = 0x00000000;                      // Reset CFGR register
}

```

```

RCC->CR      &= 0xEF7FFFFF;           // Reset CSSON and PLLON bits
RCC->PLLCFGR = 0x24003010;         // Reset PLLCFGR register
RCC->CR      &= 0xFFFFBFFFFF;        // Reset HSEBYP bit
RCC->CIR     = 0x00000000;          // Disable all interrupts

// Activate the ART + 7ws (OK for 216-MHz)

FLASH->ACR   = FLASH_ACR_ARTEN    // Activate the ART
| FLASH_ACR_PRFTEN
| FLASH_ACR_LATENCY_7WS;          // Prefetch enable
                                  // 7 wait states

// For f(ck in) = 16-MHz
// f(out) = f(vco) / P            f(out) = 216-MHz, P = 2    ---> f(vco) = 432-MHz
// f(vco) = f(ck in) * (N/M)       N/M = 432/16 = 27        ---> N = 432, M = 16
// f(usb) = f(vco) / Q            Q = 9                      ---> f(usb) = 48-MHz

RCC->PLLCFGR = 0x20000000          // Init the register
| (9<<24)
| (1<<22)
| (0<<16)
| (432<<6)
| (16<<0);                     // Q = 9
                                  // HSE as a PLL source
                                  // P = 2
                                  // N = 432
                                  // M = 16

RCC->CFGR    = (0<<30)           // Sysclk selected
| (0<<27)                      // MCO2 (no division)
| (4<<24)                      // MCO1 216/2 = 108-MHz
| (0<<23)                      // I2S uses a PLL
| (3<<21)                      // MCO1, PLL source
| (0<<16)                      // No RTC clock
| (4<<13)                      // APB2 bus @ 216/2 = 108-MHz
| (5<<10)                      // APB1 bus @ 216/4 = 54-MHz
| (0<<4)                       // HPRE bus @ 216-MHz
| (2<<0);                      // PLL used as a Sysclk

RCC->CR      &= 0xFFFFFFFF;        //
RCC->CR     |= (1<<24);          // PLL on

RCC->APB1ENR |= RCC_APB1ENR_PWRREN; // 
PWR->CR1     |= PWR_CR_ODEN;       // Activate the overdrive

while (!(RCC->CR & (1<<25)));
}

```

```
/*
 * \brief _FMC_Configuration
 *
 * - FMC configuration
 *
 */
static void _FMC_Configuration(void) {
    volatile uint32_t i, *memory;

    RCC->AHB3ENR |= RCC_AHB3ENR_FSMCEN; // Turn on the FSMCEN

    for (i = 0; i < 1000; i++);

    // FMC bank 1 & CE1 configuration in the asynchronous mode
    // - Address/Data non multiplexed
    // - FTDI (200-ns)
    // - 8 bits
    // - Write operations are enabled for the bank by the FMC (default after reset)
    // - Write operations are always performed in asynchronous mode

    FMC_Bank1->BCR1 = FMC_BCR_WREN; // Enable the write enable

    FMC_Bank1->BTR1 = (0<<28) // Access mode A
                        | (0<<24) // DATLAT (do not care)
                        | (0<<20) // CLKDIV (do not care)
                        | (0<<16) // BUSTURN of 0 clks
                        | (35<<8) // DATAST of 35 clks
                        | (0<<4) // ADDHLD (do not care)
                        | (7<<0); // ADDSET 7 of clks

    FMC_Bank1->BCR1 |= FMC_BCR_MBKEN; // Memory bank enable

    // FMC bank 1 & CE3 configuration in the asynchronous mode
    // - Address/Data non multiplexed
    // - EXTENSION (200-ns)
    // - 8 bits
    // - Write operations are enabled for the bank by the FMC (default after reset)
    // - Write operations are always performed in asynchronous mode

    FMC_Bank1->BCR3 = FMC_BCR_WREN; // Enable the write enable

    FMC_Bank1->BTR3 = (0<<28) // Access mode A
                        | (0<<24) // DATLAT (do not care)
                        | (0<<20) // CLKDIV (do not care)
                        | (0<<16) // BUSTURN of 0 clks
                        | (35<<8) // DATAST of 35 clks
                        | (0<<4) // ADDHLD (do not care)
                        | (7<<0); // ADDSET 7 of clks

    FMC_Bank1->BCR3 |= FMC_BCR_MBKEN; // Memory bank enable
```

```

// FMC bank 5-6 & CE1 configuration in the synchronous mode
// - SDRAM is a IS42S16400J-7 speed grade, connected to bank 2 (0xD0000000)
// Some bits in SDCR[1] are don't care, and the have to be set in SDCR[0],
// they aren't just don't care, the controller will fail if they aren't at 0

FMC_Bank5_6->SDCR[0] = (1<<12)                                // Burst
| (2<<10);                                         // SDRAM runs @ 108-MHz

FMC_Bank5_6->SDCR[1] = (0<<9)                                // Write allowed
| (2<<7)                                         // CAS latency 2 cycles
| (1<<6)                                         // 4 internal banks
| (1<<4)                                         // 16-bit data bus
| (1<<2)                                         // 12-bit row address
| (0<<0);                                         // 8-bit column address

// One SDRAM clock cycle is 1/108-MHz = 9.25-ns
// Some bits in SDTR[1] are don't care, and the have to be set in SDTR[0],
// they aren't just don't care, the controller will fail if they aren't at 0

FMC_Bank5_6->SDTR[0] = ((2-1)<<20)                                // 2 cycle TRP (18.5-ns > 15-ns)
| ((7-1)<<12);                                         // 7 cycle TRC (64.75 > 63-ns)

FMC_Bank5_6->SDTR[1] = ((2-1)<<24)                                // 2 cycle TRCD (18.5 > 15-ns)
| ((3-1)<<16)                                         // 3 cycle TWR
| ((5-1)<<8)                                         // 5 cycle TRAS (46.25 > 42-ns)
| ((8-1)<<4)                                         // 8 cycle TXSR (74 > 70-ns)
| ((2-1)<<0);                                         // 2 cycle TMRD

FMC_Bank5_6->SDCMR = (1<<3)                                // Command on bank 2
| (1<<0);                                         // MODE = 001, Clock Cnf Enable

while (FMC_Bank5_6->SDSR & FMC_SDSR_BUSY);           // Waiting for the ready

// ST and SDRAM datasheet agree a delay > 100-us

for (i = 0; i < 10000000; i++);

FMC_Bank5_6->SDCMR = (1<<3)                                // Command on bank 2
| (2<<0);                                         // MODE = 010, PALL

while (FMC_Bank5_6->SDSR & FMC_SDSR_BUSY);           // Waiting for the ready

FMC_Bank5_6->SDCMR = ((2-1)<<5)                                // NRFS = 2
| (1<<3)                                         // Command on bank 2
| (3<<0);                                         // MODE = 011, Auto-refresh

while (FMC_Bank5_6->SDSR & FMC_SDSR_BUSY);           // Waiting for the ready

```

```

// SDRAM mode register
// Mode: 11 10 09 08    07 06 05 04    03 02 01 00
//       - - 1 0      0 0 1 0      0 0 0 0
//
// M9      = 1      Single location access
// M8 - M7 = 00     Standard operation
// M6 - M4 = 010   Cas latency 2
// M3      = 0      Sequential
// M2 - M0 = 011   Burst length 8

FMC_Bank5_6->SDCMR = (0x220<<9)           // MRD = 0x220
                           | (1<<3)           // Command on bank 2
                           | (4<<0);          // MODE = 100, Load Mode Reg

while (FMC_Bank5_6->SDSR & FMC_SDSR_BUSY);    // Waiting for the ready

// 64-ms/4096 = 15.625-us
// 15.625-us * 108-MHz = 1688-20 = 1668

FMC_Bank5_6->SDRTTR = (1668<<1);          // Refresh timer count
while (FMC_Bank5_6->SDSR & FMC_SDSR_BUSY);    // Waiting for the ready

// New attributes for the SDRAM area (0xD0000000)
// Change the attribute only in the executable area

MPU->CTRL = 0x00000000;                      // Disable the MPU
MPU->RNR  = 0x00000000;                      // Region 0
MPU->RBAR = 0xD0000000 | (1<<4) | (0<<0); // Address, valid & region 0
MPU->RASR = (0<<28)                         // Instruction fetches enabled
                           | (3<<24)           // Full access
                           | (0<<19)           // TEX: 000 Memory attribute
                           | (1<<18)           // S: 1
                           | (1<<17)           // C: 1
                           | (0<<16)           // B: 0 sharable, Normal,
                           | (0<<8);           // Outer and inner
                           | (22<<1);          // write-through.
                           | (1<<0);           // No write allocate
                           | (0<<8);           // Corresponding sub-region
                           | (0<<8);           // is enabled
                           | (1<<1);           // 8-MB
                           | (1<<0);           // Region enable

MPU->CTRL = (1<<2)                          // Enable the usage of
                           | (0<<1);           // all the default map
                           | (1<<0);           // MPU is disabled during
                           | (1<<0);           // the fault
                           | (1<<0);           // MPU enabled

```

```
MEMO_SYNC_BARRIER;
DATA_SYNC_BARRIER;
INST_SYNC_BARRIER;

// Enable branch prediction
// Normally not necessary (always on)

    SCB->CCR |= (1<<18);
    DATA_SYNC_BARRIER;

// Clear the SDRAM

    memory = (volatile uint32_t *)&_stEXRAM;
    for (i = 0; i < (uint32_t)&_lnEXRAM; i += 4)
        *memory++ = 0x00000000;
}

/*
 * \brief _CACHE_Enable
 *
 * - Enable the L1 instruction & the data caches
 *
 */
static void _CACHE_Enable(void) {

    #if (defined(__CACHE_I__))
    cache_I_Enable();
    #endif

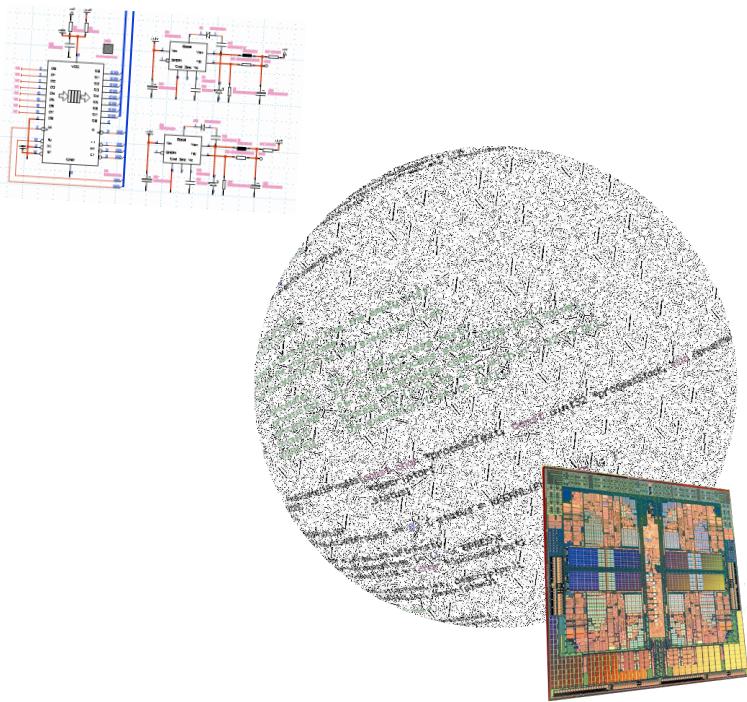
    #if (defined(__CACHE_D__))
    cache_D_Enable();
    #endif
}

#include <model_cache.cm>
```

v 3.2

A. Annex

A complete application



A.1. A full application

The best way to understand how to work with μKOS is probably to create a full application that involves all the different topics, from the low level to the host application. A simple (and minimalist) application that allows to cover all these topics could be a temperature acquisition system (from the hardware to all the associated pieces of software).

A.1.1. Main specifications

Temperature manager (tmp0):

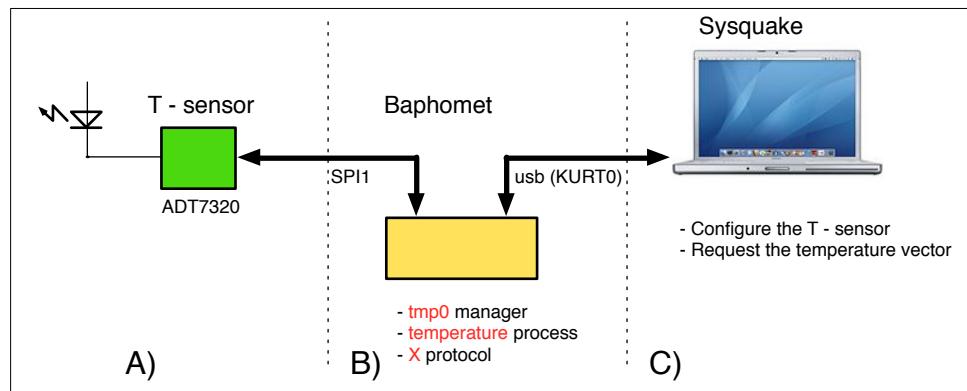
- tmp0_reserve Reserve the device.
- tmp0_release Release the device.
- tmp0_getTemperature Get the temperature.
- tmp0_setTemperature Set the temperature.

Background process (temperature):

- 1 conversions (acquisition) per 200-ms; publish a vector containing the last 128 measurements.

Protocol (X):

- Send: X Receive: x,v0,v1,v2..,v127.



A.2. The Baphomet hardware (A)

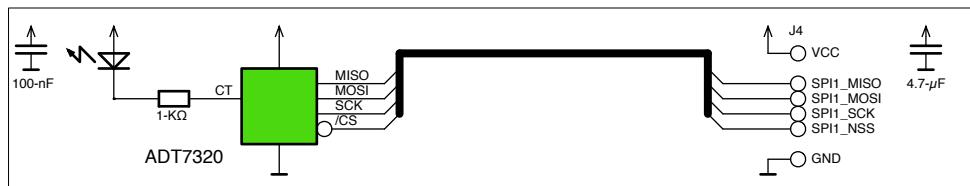
A.2.1. The temperature sensor

The selected temperature sensor is the ADT7320 from Analog Devices. It contains an internal band gap reference, a temperature sensor, and a 16-bit analog-to-digital converter to monitor and digitize the temperature to a resolution of 0.0078°C. The ADC resolution is a user-programmable mode that can be changed through the SPI interface.

A LED is connected to the sensor in order to indicate when the temperature reaches a programmed value.

The ADT7320 is guaranteed to operate over supply voltages from 2.7-V to 5.5-V. Operating at 3.3-V, the average supply current is typically 210- μ A. The ADT7320 has a shutdown mode that powers down the device and offers a shutdown current of typically 2.0- μ A at 3.3-V. The ADT7320 is rated for operations over the -40-°C to +150-°C temperature range.

The CT pin is an open-drain output (that controls a LED) that becomes active when the temperature exceeds a programmable critical temperature limit.



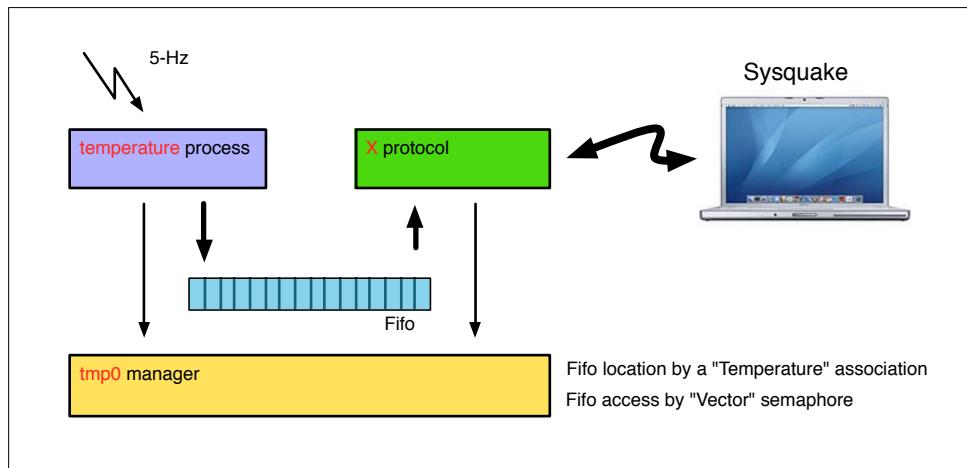
A.3. The Baphomet software (B)

For this application we need to create three specific programs inside the μKOS framework.

- The manager **tmp0**.
- The background process **temperature**.
- The protocol **X**.

It is necessary to reserve their specific identifiers in the **modules.h**. So, here are the selected identifiers for the manager, for the process and for the protocol.

```
#define      KTMPONUM          (( '0'<<8) +'4')      // Tmp0 manager (demo)
#define      KTEMPERATURENUM    (( 'X'<<8) +'T')      // Temperature process (demo)
#define      KXNUM               (( 'X'<<8) +'0')      // X protocol (demo)
```



A.3.1. “tmp0” manager system calls



tmp0 System Calls	
tmp0_reserve	Reserve the peripheral
tmp0_release	Release the peripheral
tmp0_getTemperature	Get the temperature
tmp0_setTemperature	Set the temperature

int32_t tmp0_reserve(uint8_t mode)

Reserve the peripheral.

input:	mode	KDEVALL	For read and write operations
output:	status	KTMPONOERR	The peripheral is reserved
		KTMAPCHBSY	The peripheral is busy
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t      status;  
...  
status = tmp0_reserve(KDEVALL);  
while (tmp0_reserve(KDEVALL) == KTMAPCHBSY) {  
    kern_switchFast();  
}  
status = tmp0_xyz();  
status = tmp0_release(KDEVALL);
```

This can be rewritten:

```
RESERVE(TMP0, KDEVALL);  
status = tmp0_xyz();  
RELEASE(TMP0, KDEVALL);
```

```
int32_t tmp0_release(uint8_t mode)
```

Release the peripheral.

input:	mode	KDEVALL	For read and write operations
output:	status	KTMPONOERR	OK
reentrancy:	NO		Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
int32_t status;  
...  
status = tmp0_release(KDEVALL);
```

```
int32_t tmp0_getTemperature(float64_t *temperature)
```

Get the temperature in degrees [C].

input:	*temperature	Ptr on the temperature
output:	status	KTMPONOERR
reentrancy:	NO	OK
		Use RESERVE/RELEASE macros
		to be thread-safe

Call example in C:

```
float64_t temperature;
int32_t status;
...
status = tmp0_getTemperature(&temperature);
```

```
int32_t tmp0_setTemperature(float64_t temperature)
```

Set the temperature in degrees [C]-

input:	temperature	The temperature
output:	status	OK
reentrancy:	NO	Use RESERVE/RELEASE macros to be thread-safe

Call example in C:

```
float64_t    temperature = 26.54;  
int32_t      status;  
...  
status = tmp0_setTemperature(temperature);
```

```
/*
; tmp0.
; =====

;-----
; Author:      Franz Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 13              $: Revision of last commit
; $Date::: 2013-11-24 19:28:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        tmp0 manager.
;
;   (c) 1992-2017, Franz Edo.
; -----
;
;   Franz Edo.          _____/_____\_____
;   5-Route de Cheseaux  / / / ,< / / / \ \ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \ \ \ / /
;                           \_,/_\_|_\____//____/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#ifndef      __TMP0__
#define      __TMP0__

#include    <modules.h>

#define      KTMPOMAN    (KTPONUM<<8)
#define      KTMPOERR     ((KIDPERI<<24) | KTPOMAN)
```

```
// Prototypes
// -------

#ifndef __cplusplus
extern "C" {
#endif

extern int32_t tmp0_reserve(uint8_t mode);
extern int32_t tmp0_release(uint8_t mode);
extern int32_t tmp0_getTemperature(float64_t *temperature);
extern int32_t tmp0_setTemperature(float64_t temperature);

#ifndef __cplusplus
}
#endif

// tmp0 manager errors & status
// -------

#define KTMP0NOERR 0 // No error
#define KTMP0SYCNA -(KTMP0ERR+1) // System call not available
#define KTMP0GEERR -(KTMP0ERR+2) // General error
#define KTMP0CHBSY -(KTMP0ERR+3) // The manager is busy
#endif
```

```
/*
; tmp0.
; =====

;-----
; Author:      Franz Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 13              $: Revision of last commit
; $Date::: 2013-11-24 19:28:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        tmp0 manager.
;
;   (c) 1992-2017, Franz Edo.
; -----
;
;   Franz Edo.          _____/_____\_____
;   5-Route de Cheseaux  / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \_\_ / /
;                           \_,/_\_|_\_\_//\_\_/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
#include      <uKOS.h>

static bool_t vReserved  = FALSE;

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "tmp0           tmp0 manager.
                                         (c) EFr-2017";

LOC_CONST_STRG(aStrHelp[]) =
    "tmp0 manager\n"
    "=====\\n\\n"

    "This manager ...\\n\\n";

// Prototypes
// =====

extern void          stub_tmp0_init(void);
extern int32_t        stub_tmp0_getTemperature(float64_t *temperature);
extern int32_t        stub_tmp0_setTemperature(float64_t temperature);

// Module specifications (See the modules.h)
// =====

MODULE(Tmp0, KIDPERI, KTMP0NUM, 0, "2.0", (1<<BSHOW));

/*
 * \brief Reserve the tmp0 manager
 *
 * \param[in]      mode      Any mode
 * \param[out]     KTMP0NOERR  The manager is reserved
 * \param[out]     KTMP0CHBSY  The manager is busy
 *
 */
int32_t    tmp0_reserve(uint8_t mode) {

    _init();

    INTERRUPTION_OFF;
    if (vReserved)      {                               RETURN_INT_RESTORED(KTMP0CHBSY); }
    else                { vReserved = TRUE; RETURN_INT_RESTORED(KTMP0NOERR); }
}
```

```
/*
 * \brief Release the tmp0 manager
 *
 * \param[in]      mode      Any mode
 * \param[out]     KTMP0NOERR OK
 *
 */
int32_t      tmp0_release(uint8_t mode) {

    _init();

    INTERRUPTION_OFF;
    vReserved = FALSE;
    RETURN_INT_RESTORED(KTMP0NOERR);
}

/*
 * \brief Get the temperature
 *
 * \param[in]      *temperature Ptr on the temperature
 * \param[out]     KTMP0NOERR OK
 *
 */
int32_t      tmp0_getTemperature(float64_t *temperature) {

    _init();
    return (stub_tmp0_getTemperature(temperature));
}

/*
 * \brief Set the temperature
 *
 * \param[in]      temperature The temperature
 * \param[out]     KTMP0NOERR OK
 *
 */
int32_t      tmp0_setTemperature(float64_t temperature) {

    _init();
    return (stub_tmp0_setTemperature(temperature));
}
```

```
// Local routines
// =====

/*
 * \brief _init
 *
 * - This function initializes the manager and
 *   has to be called at least once.
 *
 */
static void _init(void) {
    static bool_t init = FALSE;

    if (!init) {
        init = TRUE;

        stub_tmp0_init();
    }
}
```

```
/*
; stub.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 13              $: Revision of last commit
; $Date::: 2013-11-24 19:28:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        stub for the "tmp0" manager module.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          _____/_____\_____
;   5-Route de Cheseaux    / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \_\_ / /
;                           \_,/_\_|_\_\_//___/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include     <uKOS.h>
```

```

#define      BSELADT7320          4           // PORT A 4, /CS of the ADT7320
#define      KNBREGIST          0x07        // Nb. of registers
#define      KWRITE              0x00        // Write bit
#define      KREAD               0x40        // Read bit

#define      REGSTATUS           (0x0<<3)   // Register status (R) 8
#define      REGCONFIG            (0x1<<3)   // Register configuration (W) 8
#define      REGTEMPER            (0x2<<3)   // Register temperature (R) 16
#define      REGID                (0x3<<3)   // Register ID (R) 8
#define      REGTCRSET            (0x4<<3)   // Register T Crit set point (W) 16
#define      REGTHYSET             (0x5<<3)   // Register T Hyster set point (W) 8
#define      REGTHISET             (0x6<<3)   // Register T High set point (W) 16
#define      REGTLOSET             (0x7<<3)   // Register T Low set point (W) 16

// Prototypes
// =====

static uint8_t      _writeReadSPI(uint8_t data);

/*
 * \brief stub_tmp0_init
 *
 * - Initialize some specific CPU parts
 *
 */
void    stub_tmp0_init(void) {

    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;           // Turn on the SPI1

    // Initialize the SPI1 Master, POL0 = CPHAO = 1, f42 / 32 (1.3-MBit/s)

    GPIOA->ODR |= (1<<BSELADT7320);                  // _/ the NSS chip select
    SPI1->CR1  = SPI_CR1_MSTR;                      // Master on
    | SPI_CR1_CPHA;                                // PHA 1
    | SPI_CR1_CPOL;                                // POL 1
    | SPI_CR1_BR_2;                                 // f42 / 32
    SPI1->CR2  |= SPI_CR2_SSOE;                     //
    SPI1->CR1  |= SPI_CR1_SPE;                      // Enable the SPI1

    // Reset of the serial interface of the ADT7320

    GPIOA->ODR &= ~(1<<BSELADT7320);                // _/ the NSS chip select
    _writeReadSPI(0xFF);                            //
    _writeReadSPI(0xFF);                            //
    _writeReadSPI(0xFF);                            //
    _writeReadSPI(0xFF);                            // At least 32 bits @ 1
    GPIOA->ODR |= (1<<BSELADT7320);                // _/ the NSS chip select
    kern_suspendProcess(1);                         //

```

```
// Initialize the ADT7320

    GPIOA->ODR &= ~(1<<BSELADT7320);
    _writeReadSPI(REGCONFIG | KWRITE);
    _writeReadSPI(0x90);
    GPIOA->ODR |= (1<<BSELADT7320);
    kern_suspendProcess(1000);
                                // \_ the NSS chip select
                                // REGCONFIG 10010000
                                // _/ the NSS chip select
                                // waiting for the end of
                                // the first conversion
}

/*
 * \brief stub_tmp0_getTemperature
 *
 * - Get the temperature
 *
 */
int32_t      stub_tmp0_getTemperature(float64_t *temperature) {
    uint8_t          msb, lsb;
    int16_t         value;

    GPIOA->ODR &= ~(1<<BSELADT7320);
    _writeReadSPI(REGTEMPER | KREAD);
    msb = _writeReadSPI(0xFF);
    lsb = _writeReadSPI(0xFF);

    value = (int16_t)((msb<<8) | lsb);
                                // register value_16

    // Scale the result

    *temperature = 0.0078 * (float64_t)value;
                                // 
    GPIOA->ODR |= (1<<BSELADT7320);
                                // _/ the NSS chip select
    return (KTMP0NOERR);
}
```

```
/*
 * \brief stub_tmp0_setTemperature
 *
 * - Set the temperature
 *
 */
int32_t      stub_tmp0_setTemperature(float64_t temperature) {
    uint8_t      msb, lsb;
    uint16_t     value;

    value = (int16_t)(temperature / 0.0078);           // 
    msb = (uint8_t)(value>>8);                      // 
    lsb = (uint8_t)(value);                            // 

    GPIOA->ODR &= ~(1<<BSELADT7320);                // \_ the NSS chip select
    _writeReadSPI(REGTHISET | KWRITE);                  // 
    _writeReadSPI(msb);                                // register msb value
    _writeReadSPI(lsb);                                // register lsb value
    GPIOA->ODR |= (1<<BSELADT7320);                 // _/ the NSS chip select
    return (KTMP0NOERR);
}

// Local routines
// =====

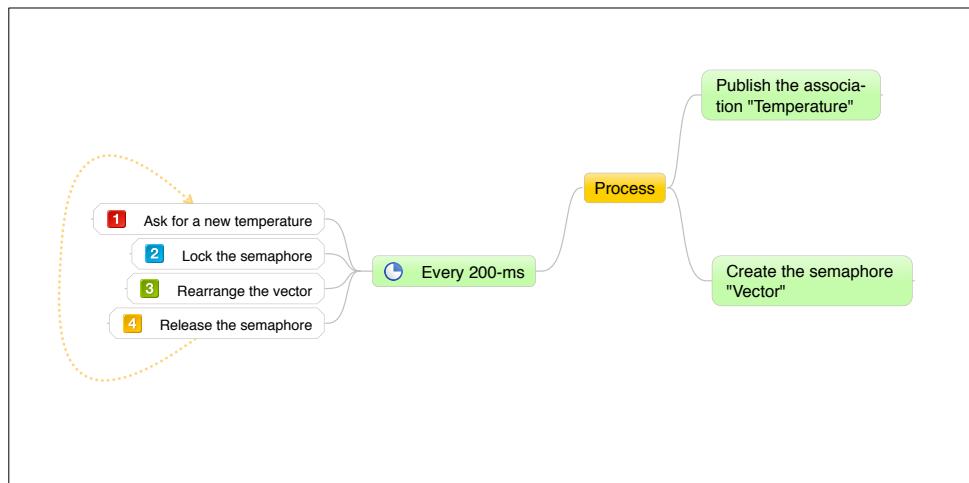
/*
 * \brief _writeReadSPI
 *
 */
static uint8_t      _writeReadSPI(uint8_t data) {
    volatile     uint16_t     dummy __attribute__ ((unused));

// Send a Byte ... and waiting for the end of the transfer

    dummy = SPI1->DR;                                // Clear the RX flag
    SPI1->DR = (uint16_t)data;                         // Send a data
    while (!(SPI1->SR & SPI_SR_RXNE));              // Waiting for the RX flag
    __asm__("nop");                                    // 
    __asm__("nop");                                    // Ensure the overlap of the CS
    return ((uint8_t)SPI1->DR);                        //
}
```

A.3.2. The background process

A simple background process is responsible to cyclically collect the temperature from the sensor and to publish a vector containing the last 128 samples. For publishing the vector we use a shared memory (association). This vector needs to be protected by a semaphore in order to manage the possible simultaneous access (from this process and from the protocol module).



In order to test the software project without having any hardware in place, the compilation flag **_SIMULE_** allows to configure the **background process** for emulating a temperature profile. To activate this possibility, just activate it before the compilation.

```
#define _SIMULE_
```

```
/*
; temperature.
; =====

;-----
; Author:      Franz Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 18              $: Revision of last commit
; $Date::: 2013-12-15 19:05:00#$: Date of last commit
;
; Project:     uKOS
; Goal:        temperature process; continuous acquisition of the temperature.
;
;   (c) 1992-2017, Franz Edo.
; -----
;
;   Franz Edo.          _____/_____\_____
;   5-Route de Cheseaux    / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \_\_ / /
;                           \_,/_\_|_\_\_//___/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
#include      <uKOS.h>
#include      <stdlib.h>

#define      __SIMULE__

#define      KTIMEACQ     200          // 200-ms
#define      KNBSAMPLES   128          // Nb. of samples (size of the vector)

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "temperature  temperature acquisition process           (c) EFr-2017";
LOC_CONST_STRG(aStrHelp[]) =
    "temperature process\n"
"=====\\n\\n"

    "Acquisition of the temperature\\n\\n";

LOC_CONST_STRG(aStrIden[]) = "Process_temperature";
LOC_CONST_STRG(aStrText[]) = "Process temperature: temp acquisition.      (c) EFr-2017";

// Prototypes
// =====

static int32_t      prgm(uint32_t argc, char_t *argv[]);
static void         _process(void);

// Module specifications (See the modules.h)
// =====

MODULE(Temperature, KIDPROCESS, KTEMPERATURENUM, prgm, "1.0", (1<<BSHOW));

/*
 * \brief Main entry point
 *
 */
static int32_t      prgm(uint32_t argc, char_t *argv[]) {
    volatile      proc_t        *process;

    PROC_SUPV(0, vSpecification, aStrText, KSZSTACKMIN, \
              _process, aStrIden, KDEF0, 31, 0, KPROCNORMAL);

    if (kern_createProcess(&vSpecification, &process) != KKERNNOERR)
        exit(EXIT_FAILURE);
    return (EXIT_SUCCESS_OS);
}
```

```
/*
 * \brief _process
 *
 * - Temperature acquisitions (5-Hz)
 *
 */
static void _process(const void *argument) {
    volatile astp_t      *association;
    volatile sema_t      *semaphore;
    static volatile int16_t i, vTemperature[KNBSAMPLES];

#if (defined(__SIMULE__))
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100, \
    20.23*100, 20.52*100, 21.23*100, 21.87*100, \
    22.21*100, 22.67*100, 23.12*100, 23.67*100, \
    23.78*100, 23.34*100, 22.76*100, 22.09*100, \
    21.56*100, 21.14*100, 20.55*100, 20.03*100 };
```

#else

float64_t temperature;

#endif

```
// Create the association "Temperature" and the semaphore "Vector"

pack.oGeneral = (void *)&vTemperature[0];
pack.oSize = sizeof(vTemperature);

if (kern_createAssociation("Temperature", &association) != KKERNNOERR)
    exit(EXIT_PANIC);
if (kern_publishAssociation(association, &pack) != KKERNNOERR)
    exit(EXIT_PANIC);
if (kern_createMutxSemaphore("Vector", &semaphore) != KKERNNOERR)
    exit(EXIT_PANIC);

RESERVE(TMP0, KDEVALL);

while (TRUE) {
    kern_suspendProcess(KTIMEACQ);

// Waiting for the semaphore
// Rearrange the vector & store the new value inside the FIFO
// Release the semaphore
// If we use the __SIMULE__ flag (emulation of the temperature acquisition)
// then, a random noise (-1 .. +1 degree) is added

    kern_lockSemaphore(semaphore, -1);

    #if (defined(__SIMULE__))
    for (i = 0; i < KNBSAMPLES; i++)
        vTemperature[i] = aSimule[i] + ((int16_t)(rand())>>8);

    #else
    for (i = 0; i < (KNBSAMPLES-1); i++)
        vTemperature[KNBSAMPLES-1-i] = vTemperature[KNBSAMPLES-2-i];

    tmp0_getTemperature(&temperature);
    vTemperature[0] = (int16_t)(temperature * 100.0);
    #endif

    kern_unlockSemaphore(semaphore);
}
}
```

A.3.3. The protocol

The protocol module X is responsible to react from the messages coming from a host computer. For our application, it allows to read the temperature vector (in ASCII).

```
/*
; X.
; ==

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 13              $: Revision of last commit
; $Date::: 2013-11-24 19:28:43#$: Date of last commit
;
; Project:     uKOS
; Goal:        Control the temperature.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          / \ / \ / \ / \
;   5-Route de Cheseaux  / / / , < / / / / \_ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / / \_ / /
;                           \_,/_ | \_ / / \_ / /
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
; -----
*/

```

```
#include      <uKOS.h>

#define       KNBSAMPLES    128           // Nb. of samples (size of the vector)

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "X                 Control the temperature.                                (c) EFr-2017";
LOC_CONST_STRG(aStrHelp[]) =
    "Control the temperature\n"
    "=====\\n\\n"

    "This protocol module allows to operate on the xtmp\\n"
    "temperature manager\\n\\n"

    "Input format: X\\n"
    "Output format: x,t0,t1,t...t127\\n\\n";

// Prototypes
// =====

static int32_t      prgm(uint32_t argc, char_t *argv[]);

// Module specifications (See the modules.h)
// =====

MODULE(X, KIDPROTOCOL, KXNUM, prgm, "1.0", ((1<<BSHOW) | (1<<BEXECONSOLE)));

/*
 * \brief Main entry point
 *
 */
static int32_t      prgm(uint32_t argc, char_t *argv[]) {
    uint16_t      *temperature, i;
    volatile astp_t      *association;
    volatile sema_t      *semaphore;

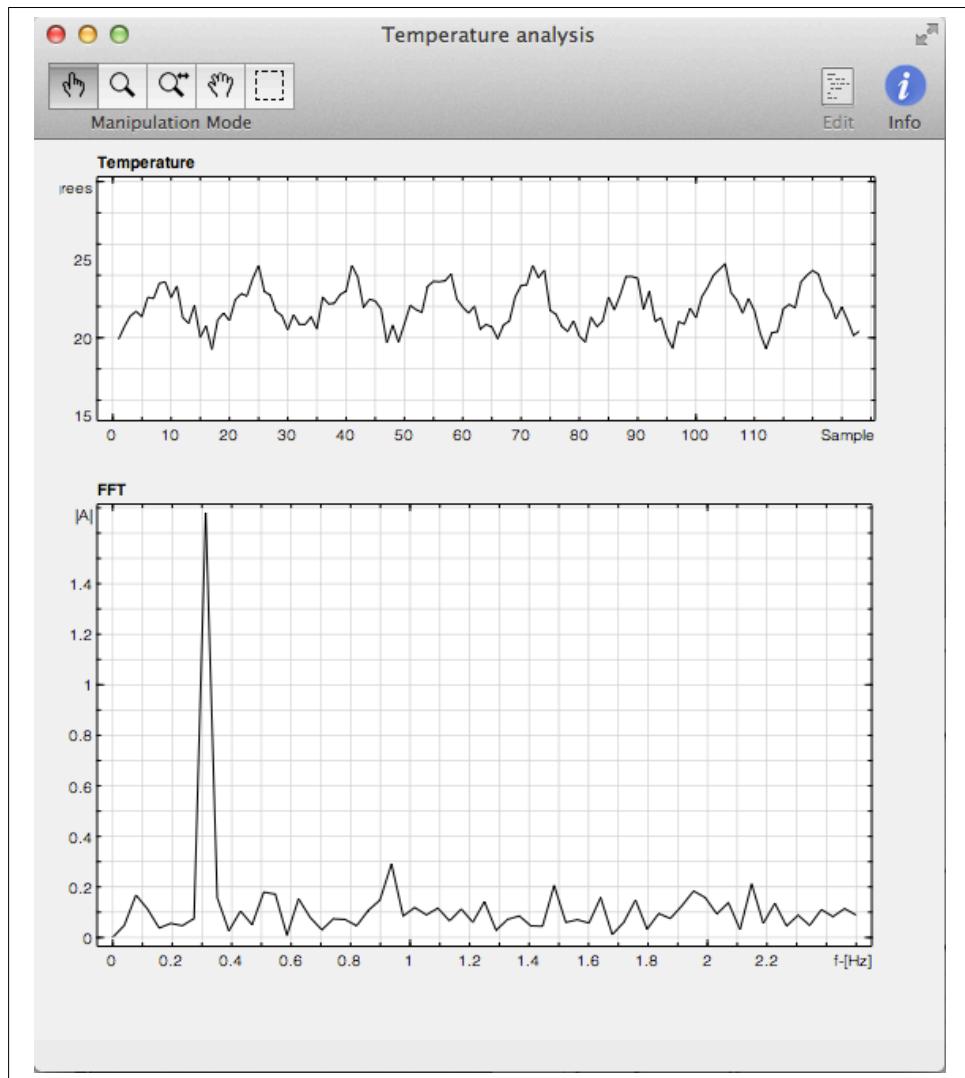
    enum { KERRNOT, KERRASS, KERRSEM } error = KERRNOT;

    if (kern_getAssociationById("Temperature", &association) != KKERNNOERR)
        error = KERRASS;
    if (kern_findAssociation(association, (void *)&temperature) != KKERNNOERR)
        error = KERRASS;
    if (kern_getSemaphoreById("Vector", &semaphore) != KKERNNOERR)
        error = KERRSEM;
```

```
switch (error) {  
    case KERRNOT: {  
        kern_lockSemaphore(semaphore, -1);  
        iotx_printf(KSYST, "x,");  
        for (i = 0; i < (KNBSAMPLES-1); i++)  
            iotx_printf(KSYST, "%d,", temperature[i]);  
  
        iotx_printf(KSYST, "%d\n", temperature[127]);  
  
        kern_unlockSemaphore(semaphore);  
        return (EXIT_SUCCESS_OS);  
    }  
    case KERRASS: {  
        iotx_printf(KSYST, "z,Association error.\n");  
        return (EXIT_FAILURE);  
    }  
    case KERRSEM: {  
        iotx_printf(KSYST, "z,Semaphore error.\n");  
        return (EXIT_FAILURE);  
    }  
    default: return (EXIT_FAILURE);  
}
```

A.4. The Sysquake software (C)

Sysquake is the perfect **VST** (Visualization Software Tool) usable to manipulate the data coming from the hardware target. Cyclically, it asks for a new temperature vector and then, it can manipulate and display it. Another **VST** like Matlab, LabView or Mathematica can also be used for this purpose.



```
//-----  
// Author: Franzi Edo. The 2014-01-01  
// Modifs:  
//  
// SVN:  
// $Author:: efr           $: Author of last commit  
// $Rev::: 8              $: Revision of last commit  
// $Date:: 2013-11-24 19:27:53#$: Date of last commit  
//  
// Project: uKOS  
// Goal: Read a 128 element vector (temperature).  
//        Display the temperature vector  
//        Display the FFT of the temperature vector  
//  
// (c) 1992-2017, Franzi Edo.  
// -----  
// Franzi Edo.          / \ / / / \ / \ / / / /  
// 5-Route de Cheseaux   / / / , < / / / \ \ \ \ \ /  
// CH 1400 Cheseaux-Noréaz / / / / | / / / / \ / / /  
//                           \ \ \ \ / | \ \ \ \ / / / /  
// edo.franzi@ukos.ch  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU Affero General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU Affero General Public License for more details.  
//  
// You should have received a copy of the GNU Affero General Public License  
// along with this program. If not, see <http://www.gnu.org/licenses/>.  
//-----  
  
title "Temperature analysis"  
  
version  
{@  
Version 1.0, 01 January 2016  
  
Copyright 2016, uKOS www.ukos.ch  
{@}
```

```
help
{@
This SQ file allows to display the temperature vector.
```

1. Connect the board to the USB port of your computer
2. Be sure that the USB port is not already in use
3. Launch the Sysquake executable X.sqd

Important: do not change the device selection while the application is running. Before removing the USB cable, it is mandatory to quit Sysquake or any applications having the control of the USB port!

```
@}
```

```
extension "serialdev"

variable      fd, devList, devIndex
variable      temperature

init          (devList, devIndex) = initDevList
init          (fd) = init(devList, devIndex)

idle         (temperature) = acquisition(fd)

terminate    terminate(fd)
beforedump   terminate(fd)

figure "Temperature"
        draw drawTemperature(temperature)

figure "FFT"
        draw drawFFT(temperature)

restore      (devList, devIndex) = initDevList
restore      (fd) = init(devList, devIndex)
```

```
functions
{@

define KCOMMAND_X      = 0x58          // X command
define KNBSAMPLES      = 128           // Number of samples
define KNBSAMPLESP2    = 128           // Number of samples (upper pow of 2)
define KFS              = 5              // Sampling frequency (5-Hz)
define KTS              = 1/KFS         // Sampling time (200-ms)

// Open the device and prepare for displaying the data
// _____

function (fd) = init(devList, devIndex)
    fd = serialdevopen(devList{devIndex}, ...
        serialdevset('BPS', 460800, 'TextMode', true, 'Timeout', 2));
    fflush(fd);

    subplots(['Temperature\nFFT']);

// Looking for a Baphomet device (standard console on the device C)
// _____

function (devList, devIndex) = initDevList
    devList = serialdevname;
    devList = devList(cellfun(@(name) ...
        ~isempty(strfind(name, 'usbserial'))), devList));

    devIndex = find(cellfun(@(name) name(end) == 'C', devList));
    if ~isempty(devIndex)
        devIndex = devIndex(1);
    end

// Terminate
// _____

// Before closing the channel, we have to be sure
// that it is empty!!

function terminate(fd)
    try
        while true
            dummy = fread(fd, 1000, '*uint8');
        end
    end
    fflush(fd);
    fclose(fd);
```

```
// Compute ...
// =====

// acquisition loop
// ----

function (temperature) = acquisition(fd)
    fflush(fd);
    fprintf(fd, 'x\n');
    answer = fgetl(fd);

    data = sscanf(answer, 'x,%s');
    temperature = sscanf(data, '%d,');
    temperature = (temperature'/100);

// Display the temperature
// ----

function drawTemperature(temperature)
    m = min(temperature);
    M = max(temperature);
    scale([0, KNBSAMPLES, 15, 30]);
    plot(temperature);
    label Sample Degrees;

// Display the FFT
// ----

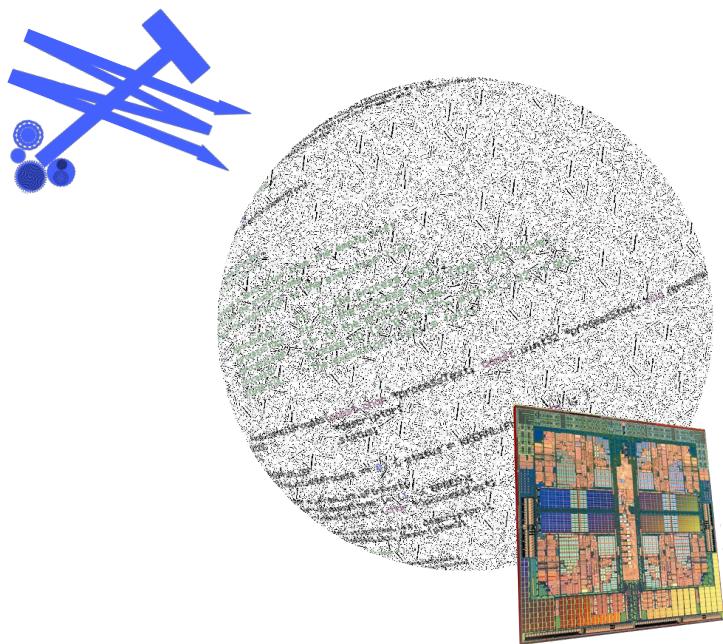
function drawFFT(temperature)
    frequency = (KFS/2)*linspace(0, 1, (KNBSAMPLESP2/2)+1);
    amplitude = fft(temperature, KNBSAMPLESP2)/KNBSAMPLES;
    amplitude = abs(amplitude(1:(KNBSAMPLESP2/2)+1));

    DC = amplitude(1)
    amplitude(1) = 0;
    plot(frequency, (2*amplitude));
    label f-[Hz] |A|;

@}
```

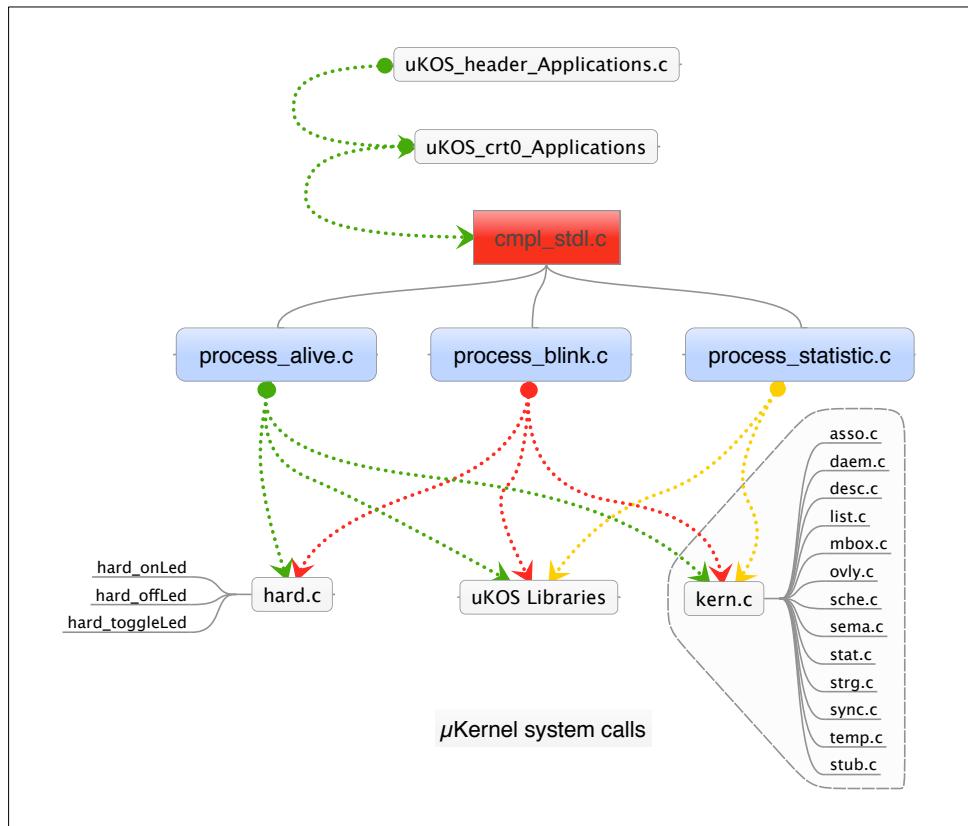
B. Annex

A standalone application using the µKernel



B.1. A standalone application

As an example here is a full standalone application (independent of the μKOS system). This example shows how to use the μKernel outside the μKOS context.



B.1.1. Building phase

Open a terminal session and:

```
cd ${PATH_UKOS_KERNEL}/Applications/uKOS_Apps/cmpl_std1/Discovery_M4  
make
```

Inside the folder ...

```
cd ${PATH_UKOS_KERNEL}/Applications/uKOS_Apps/cmpl_std1/Discovery_M4
```

... you can find the result of the building phase. The **cmpl_std1.s3** is a Motorola S-loader file containing the full application and ready to be downloaded into the target.

B.1.2. The cmpl_std.c file

```
/*
; cmpl_stdl.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113             $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Demo of a small application using the uKOS uKernel.
;
;           Launch 3 processes:
;
;           - Process: alive
;           - Process: blink
;           - Process: display the system statistic
;
; (c) 1992-2017, Franzi Edo.
; -----
;
; Franzi Edo.          / \   / \   / \   / \
; 5-Route de Cheseaux  / / / , < / / / \ \_ \
; CH 1400 Cheseaux-Noréaz / / / / | / / / / / /
;                         \_,/_/ |_\_\_//__/
; edo.franzi@ukos.ch

;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
```

```
#include      <uKOS.h>
#include      <stdlib.h>
#include      "hard.h"

// Module strings
// =====

LOC_CONST_STRG(aStrApplication[]) =
    "cmpl_std1      Standalone application using the uKernel  (c) EFr-2017";

LOC_CONST_STRG(aStrHelp[]) =
    "This is a ROMable C application\n"
    "=====\\n\\n"

    "This user function module is a C written application.\\n\\n"

    "Input format:  cmpl_std1\\n"
    "Output format: [result]\\n\\n";

// Prototypes
// =====

extern void           install_process_alive(void);
extern void           install_process_blink(void);
extern void           install_process_statistic(void);
static void           _myRoutine(uint32_t state);

// Module specifications
// =====

MODULE(UserAppl, KIDAPPLICATION, KAPPLICATIONNUM, _start, "1.0", \
    ((1<<BSHOW) | (1<<BEXECONSOLE)));

#define      KLEDIDLE      3
```

```
/*
 * \brief main
 *
 * - Setup the exceptions
 * - Initialize the hardware
 * - Launch all the processes
 * - Kill the "main". At this moment only the launched processes are executed
 *
 */
int    main(void) {

    INTERRUPTION_OFF;

    kern_init();                                // Initialize the uKernel
    install_process_alive();                    // Install the process alive
    install_process_blink();                   // Install the process blink
    install_process_statistic();                // Install the process statistic

    kern_installCallBack(_myRoutine);           // Install the call-back
    kern_runKernel();                          // Run the uKernel

    INTERRUPTION_SET;                         // Set-up the interruptions
    return (EXIT_SUCCESS);
}

/*
 * \brief _myRoutine
 *
 * - Call back routine (called by the idle daemon)
 *
 */
static void _myRoutine(uint32_t state) {

    if (state == KINIDLE) misc_onLed(KLEDIDLE);
    else                  misc_offLed(KLEDIDLE)
}
```

```
/*
; process_alive.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113              $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Process: alive
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          _____/_____\_____
;   5-Route de Cheseaux    / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \_\_ / /
;                           \_,/_\_|_\_\_//___/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
#include      <uKOS.h>
#include      <stdlib.h>
#include      "hard.h"

#define        KTBLINK_ALIVE      1000
#define        KLEDBLINK_ALIVE    0

// Prototypes
// =====

static void _process(const void *argument);

/*
 * \brief Install & launch the process
 *
 */
void  install_process_alive(void) {
    volatile     proc_t      *process;

    LOC_CONST_STRG(aStrIden[]) = "User_Process_alive";
    LOC_CONST_STRG(aStrText[]) = "Process alive.          (c) EFr-2017";

// Specifications for the processes

    PROC_SUPV(0, vSpecification, aStrText, KSZSTACKMIN, \
              _process, aStrIden, KDEF0, 1, 0, KPROCNORMAL);

    if (kern_createProcess(&vSpecification, &process) != KKERNNOERR)
        exit(EXIT_FAILURE);
}

/*
 * \brief _process
 *
 */
static void _process(const void *argument) {

    while (TRUE) {
        kern_suspendProcess(KTBLINK_ALIVE);
        hard_toggleLed(KLEDBLINK_ALIVE);
    }
}
```

```
/*
; process_blink.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113              $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Process: blink
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          _____/_____\_____
;   5-Route de Cheseaux    / / / ,< / / / \_\ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / \_\_ / /
;                           \_,/_\_|_\_\_//___/
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
#include      <uKOS.h>
#include      <stdlib.h>
#include      "hard.h"

#define        KTBLINK_BLINK      500
#define        KLEDBLINK_BLINK    1

// Prototypes
// =====

static void _process(const void *argument);

/*
 * \brief Install & launch the process
 *
 */
void  install_process_blink(void) {
    volatile     proc_t      *process;

    LOC_CONST_STRG(aStrIden[]) = "User_Process_blink";
    LOC_CONST_STRG(aStrText[]) = "Process blink.                               (c) EFr-2017";

// Specifications for the processes

    PROC_SUPV(0, vSpecification, aStrText, KSZSTACKMIN, \
              _process, aStrIden, KDEF0, 1, 0, KPROCNORMAL);

    if (kern_createProcess(&vSpecification, &process) != KKERNNOERR)
        exit(EXIT_FAILURE);
}

/*
 * \brief _process
 *
 */
static void _process(const void *argument) {

    while (TRUE) {
        kern_suspendProcess(KTBLINK_BLINK);
        hard_toggleLed(KLEDBLINK_BLINK);
    }
}
```

```
/*
; process_statistic.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113              $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Process: display the system statistic
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          _____/ / / / \ \ / / /
;   5-Route de Cheseaux    / / / / ,< / / / / \ \ \ \
;   CH 1400 Cheseaux-Noréaz / / / / | / / / / \ \ / /
;                           \_,/_ / |_\ \ \ / / \ \ / /
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/
#include     <uKOS.h>
#include     <stdlib.h>
#include     "hard.h"
```

```
#define      KTSAMPLE     2000
#define      KLNASCBUF    24
#define      KDIGITFAC    100

// Prototypes
// =====

static void _process(void);
static void _printPar_P1(uint8_t priority, uint32_t mode, \
                        const char_t *textID, const char_t *identifierID);
static void _printPar_T1(uint32_t min, uint32_t max, uint32_t avg, \
                        float64_t pRatio);
static void _printPar_T2(uint32_t min, uint32_t max, uint32_t avg, \
                        float64_t kRatio);
static void _printPar_T3(uint32_t min, uint32_t max, uint32_t avg, \
                        float64_t kRatio);
static void _printPar_N1(uint64_t nbTimes);
static void _printPar_S1(uint64_t nbCalls);
static void _printPar_S2(uint32_t size);

typedef      struct process      process_t;

struct process {
    bool_t          oValid;           // Process valid
    spec_t          oSpecification; // Process specification
    work_t          oInternal;        // Process internal stuff
    stts_t          oStatistic;      // uKernel statistic
};

/*
 * \brief Install & launch the process
 *
 */
void  install_process_statistic(void) {
    volatile      proc_t      *process;

    LOC_CONST_STRG(aStrIden[]) = "User_Process_statistic";
    LOC_CONST_STRG(aStrText[]) = "Process statistic.           (c) EFr-2017";

// Specifications for the processes

    PROC_SUPV(0, vSpecification, aStrText, KSZSTACKMIN, \
              _process, aStrIden, KDEF0, 31, 0, KPROCNORMAL);

    if (kern_createProcess(&vSpecification, &process) != KKERNNOERR)
        exit(EXIT_FAILURE);
}
```

```
/*
 * \brief _process
 *
 */
static void  _process(const void *argument) {
    uint32_t      i;
    uint64_t      pRatio, kRatio, eRatio, totalTimeCPU;
    float64_t     pRatioF, kRatioF, eRatioF;
    uint8_t       *bufSysProcess;
    process_t     *sysProcess;
volatile    proc_t      *process;

    bufSysProcess = (uint8_t *)sysos_malloc(KNBPROCESS*sizeof(process_t));
    if (bufSysProcess == 0) exit(EXIT_FAILURE);

    while (TRUE) {
        kern_suspendProcess(KTSAMPLE);

        totalTimeCPU = 0;

        // Collect the process information usable for the statistics

        for (i = 0; i < KNBPROCESS; i++) {
            sysProcess = (process_t *)((uint32_t)bufSysProcess + \
                i*sizeof(process_t));

            sysProcess->oValid = FALSE;
            if (kern_getProcessDescNb(i, & process) == KKERNNOERR) {
                sysProcess->oValid = TRUE;
                sysProcess->oSpecification.oIdentifier = \
                    process->oSpecification.oIdentifier;
                sysProcess->oSpecification.oText      = \
                    process->oSpecification.oText;
                sysProcess->oSpecification.oPriority  = \
                    process->oSpecification.oPriority;
                sysProcess->oSpecification.oMode     = \
                    process->oSpecification.oMode;
                sysProcess->oInternal.oPriority     = \
                    process->oInternal.oPriority;
                sysProcess->oStatistic.oNbExecutions = \
                    process->oStatistic.oNbExecutions;
                sysProcess->oStatistic.oNbKernCalls  = \
                    process->oStatistic.oNbKernCalls;
                sysProcess->oStatistic.oAvStkSup    = \
                    process->oStatistic.oAvStkSup;
                sysProcess->oStatistic.oTimePMin    = \
                    process->oStatistic.oTimePMin;
                sysProcess->oStatistic.oTimePMax    = \
                    process->oStatistic.oTimePMax;
                sysProcess->oStatistic.oTimePAvg    = \
                    process->oStatistic.oTimePAvg;
            }
        }
    }
}
```

```
        process->oStatistic.oTimePAvg;           = \
sysProcess->oStatistic.oTimeKMin          = \
        process->oStatistic.oTimeKMin;         = \
sysProcess->oStatistic.oTimeKMax          = \
        process->oStatistic.oTimeKMax;         = \
sysProcess->oStatistic.oTimeKAvg          = \
        process->oStatistic.oTimeKAvg;
    }
}

// Print the information

for (i = 0; i < KNBPROCESS; i++) {
    sysProcess = (process_t *)((uint32_t)bufSysProcess + \
        i*sizeof(process_t));
    if (sysProcess->oValid == TRUE) {
        totalTimeCPU += \
            ((uint64_t)sysProcess->oStatistic.oTimePAvg + \
            (uint64_t)sysProcess->oStatistic.oTimeKAvg) * \
            sysProcess->oStatistic.oNbExecutions;
    }
}

for (i = 0; i < KNBPROCESS; i++) {
    sysProcess = (process_t *)((uint32_t)bufSysProcess + \
        i*sizeof(process_t));
    if (sysProcess->oValid == TRUE) {

// Compute the statistic in % of time CPU

        pRatio = sysProcess->oStatistic.oTimePAvg * \
            sysProcess->oStatistic.oNbExecutions * 100;
        kRatio = sysProcess->oStatistic.oTimeKAvg * \
            sysProcess->oStatistic.oNbExecutions * 100;
        eRatio = sysProcess->oStatistic.oTimeEAvg * \
            sysProcess->oStatistic.oNbExecutions * 100;

        pRatio = (float64_t)pRatio / totalTimeCPU;
        kRatio = (float64_t)kRatio / totalTimeCPU;
        eRatio = (float64_t)eRatio / totalTimeCPU;
```

```

// Print the string process

    _printPar_P1(sysProcess->oSpecification.oPriority, \
        sysProcess->oSpecification.oMode, \
        sysProcess->oSpecification.oText, \
        sysProcess->oSpecification.oIdentifier);
    _printPar_T1(sysProcess->oStatistic.oTimePMin, \
        sysProcess->oStatistic.oTimePMax, \
        sysProcess->oStatistic.oTimePAvg, pRatio);
    _printPar_T2(sysProcess->oStatistic.oTimeKMin, \
        sysProcess->oStatistic.oTimeKMax, \
        sysProcess->oStatistic.oTimeKAvg, kRatio);
    _printPar_T3(sysProcess->oStatistic.oTimeEMin, \
        sysProcess->oStatistic.oTimeEMax, \
        sysProcess->oStatistic.oTimeEAvg, eRatio);
    _printPar_N1(sysProcess->oStatistic.oNbExecutions);
    _printPar_S1(sysProcess->oStatistic.oNbKernCalls);
    _printPar_S2(sysProcess->oStatistic.oAvStkSup);
}
}

}

// Local routines
// =====

/*
 * \brief _printPar_XYZ
 */
static void _printPar_P1(uint8_t priority, uint32_t mode, \
                           const char_t *textID, const char_t *identifierID) {
    char_t      *space;

    if (mode == KUSER)  space = "User";
    else                 space = "Supervisor";

    iotx_printf(KSYST, "Process id:      %s\n", identifierID);
    iotx_printf(KSYST, "Process text:     %s\n", textID);
    iotx_printf(KSYST, "Process space:    %s\n", space);
    iotx_printf(KSYST, "Process priority: %2d\n", priority);
}

static void _printPar_T1(uint32_t min, uint32_t max, uint32_t avg, \
                           float64_t pRatio) {

    iotx_printf(KSYST, "CPU time used by the process,    min: %5dus \
        max: %5dus avg: %5dus %3d.%03d%\n", \
        min, max, avg, FLOAT_3(ratio));
}

```

```
static void _printPar_T2(uint32_t min, uint32_t max, uint32_t avg, \
                        float64_t kRatio) {

    iotx_printf(KSYST, "CPU time used by the kernel,      min: %5dus \
                  max: %5dus avg: %5dus %3d.%03d%%\n", \
                  min, max, avg, FLOAT_3(ratio));
}

static void _printPar_T3(uint32_t min, uint32_t max, uint32_t avg, \
                        float64_t eRatio) {

    iotx_printf(KSYST, "CPU time used by the exceptions, min: %5dus \
                  max: %5dus avg: %5dus %3d.%03d%%\n", \
                  min, max, avg, FLOAT_3(ratio));
}

static void _printPar_N1(uint64_t nbTimes) {

    iotx_printf(KSYST, "Nb of time that the process was scheduled:  %-lu\n", \
                nbTimes);
}

static void _printPar_S1(uint64_t nbCalls) {

    iotx_printf(KSYST, "Nb of system calls to the uKernel functions: %-lu\n", \
                nbCalls);
}

static void _printParameter_S2(uint32_t size) {

    iotx_printf(KSYST, "Available supervisor stack size:           %-d\n\n", \
                size);
}
```

B.1.3. The hard.c file

```
/*
; hard.
; =====

;-----
; Author:      Franzi Edo.  The 2006-06-28
; Modifs:
;
; SVN:
; $Author::: efr           $: Author of last commit
; $Rev::: 113             $: Revision of last commit
; $Date::: 2013-01-03 22:36:26#$: Date of last commit
;
; Project:     uKOS
; Goal:        Some basic hardware initializations.
;
;   (c) 1992-2017, Franzi Edo.
; -----
;
;   Franzi Edo.          / \ / / / \ \ / \ / \
;   5-Route de Cheseaux  / / / , < / / / / \ \ \
;   CH 1400 Cheseaux-Noréaz / / / / | / _/ / \ / /
;                           \_,/_/ |_\ \ / / \ / /
;
;   edo.franzi@ukos.ch
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU Affero General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU Affero General Public License for more details.
;
; You should have received a copy of the GNU Affero General Public License
; along with this program. If not, see <http://www.gnu.org/licenses/>.
;
;-----
*/

```

```
#include      <uKOS.h>

/*
 * \brief hard_onLed
 *
 */
void  hard_onLed(uint32_t ledNb) {

    INTERRUPTION_OFF;
    switch (ledNb & 0x7) {
        case 0: GPIOF->ODR |= (1<<BLED0); break;
        case 1: GPIOF->ODR |= (1<<BLED1); break;
        case 2: GPIOG->ODR |= (1<<BLED2); break;
        case 3: GPIOG->ODR |= (1<<BLED3); break;
        case 4: GPIOG->ODR |= (1<<BLED4); break;
        case 5: GPIOH->ODR |= (1<<BLED5); break;
        case 6: GPIOH->ODR |= (1<<BLED6); break;
        case 7: GPIOI->ODR |= (1<<BLED7); break;
    }
    INTERRUPTION_RESTORED;
}

/*
 * \brief hard_offLed
 *
 */
void  hard_offLed(uint32_t ledNb) {

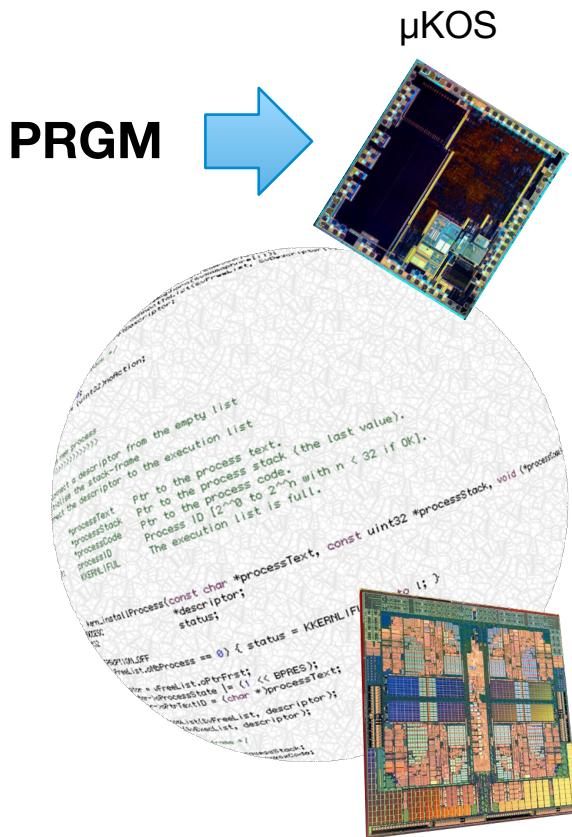
    INTERRUPTION_OFF;
    switch (ledNb & 0x7) {
        case 0: GPIOF->ODR &= (1<<BLED0); break;
        case 1: GPIOF->ODR &= (1<<BLED1); break;
        case 2: GPIOG->ODR &= (1<<BLED2); break;
        case 3: GPIOG->ODR &= (1<<BLED3); break;
        case 4: GPIOG->ODR &= (1<<BLED4); break;
        case 5: GPIOH->ODR &= (1<<BLED5); break;
        case 6: GPIOH->ODR &= (1<<BLED6); break;
        case 7: GPIOI->ODR &= (1<<BLED7); break;
    }
    INTERRUPTION_RESTORED;
}
```

```
/*
 * \brief hard_toggleLed
 *
 */
void  hard_toggleLed(uint32_t ledNb) {

    INTERRUPTION_OFF;
    switch (ledNb & 0x7) {
        case 0: GPIOF->ODR ^= (1<<BLED0); break;
        case 1: GPIOF->ODR ^= (1<<BLED1); break;
        case 2: GPIOG->ODR ^= (1<<BLED2); break;
        case 3: GPIOG->ODR ^= (1<<BLED3); break;
        case 4: GPIOG->ODR ^= (1<<BLED4); break;
        case 5: GPIOH->ODR ^= (1<<BLED5); break;
        case 6: GPIOH->ODR ^= (1<<BLED6); break;
        case 7: GPIOI->ODR ^= (1<<BLED7); break;
    }
    INTERRUPTION_RESTORED;
}
```


C. Annex

Making ROMable applications



C.1. Making ROMable applications

Making **ROMable** applications is an easy and straight forward process. All the downloadable applications can be linked in the **µKOS OS Flash**. A ROMable application could be approximately 2 to 4 times faster than a downloadable application (of course, this is an hardware dependent factor). For **STM32** cortex CPUs the measured factor is close to 4 (this is mainly due to the fact that the internal RAM/Flash have an access time of 0 wait states).

C.2. How to proceed

From a downloadable application (i.e. see the application **sqe_kalman** located in the Applications folder) here are the necessary rule/modification steps:

- Programming **rules**.
- Adaptation of the **application.c**.
- Adaptation of the main µKOS **makefile**.

C.2.1. Programming rules

To take advantage from the hardware possibilities of the **STM32** cortex CPUs (mainly the high speed of the internal RAM/Flash), here are some suggestions:

Use “const” for the array of non-modifiable data

The size of the Flash memory is much bigger than the internal RAM one. For the constant arrays, use the Flash (same speed as the internal RAM).

Instead of writing ...

```
static uint8_t      aWeightSet[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
... write  
const uint8_t      aWeightSet[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Use “syos_malloc” for reserving large data arrays

The size of the internal RAM is limited and does not allow to contain large data arrays. Large data arrays should be located inside the external memory (bigger size but lower speed). Here a trade-off between the memory size and the memory speed has to be taken in account. If the demanded size is limited, the internal reservation should be considered.

Instead of writing ...

```
static uint8_t      vImage[752*480];  
static uint8_t      vXProjection[752];  
static uint8_t      vYProjection[480];  
... write  
static uint8_t      *vImage;  
static uint8_t      *vXProjection;  
static uint8_t      *vYProjection;  
...  
vImage      = (uint8_t *)syos_malloc(KEXTERNAL, 752*480);  
vXProjection = (uint8_t *)syos_malloc(KINTERNAL, 752);  
vYProjection = (uint8_t *)syos_malloc(KINTERNAL, 480);
```

C.2.2. Adaptation of the application.c

```
...
// Prototypes
// =====

#if (defined(__ROMABLE__))
static int32_t prgm(uint32_t argc, char_t *argv[]);
#endif

// Module specifications (See the modules.h)
// =====

#if (defined(__ROMABLE__))
MODULE(UserApp1, KIDTOOL, KROMABLE0, prgm, "1.0", \
      ((1<<BSHOW) | (1<<BEXECONSOLE)));
#endif

#else
MODULE(UserApp1, KIDAPPLICATION, KAPPLICATIONNUM, _start, "1.0", \
      ((1<<BSHOW) | (1<<BEXECONSOLE)));
#endif
...
...
/*
 * \brief main
 *
 * - Initialize the used libraries
 * - Launch all the processes
 * - Kill the "main". At this moment only the launched processes are executed
 */
#endif
#if (defined(__ROMABLE__))
static int32_t prgm(uint32_t argc, char_t *argv[]) {

#else
int main(void) {
#endif

...
    return (EXIT_SUCCESS_OS);
}
```

C.2.3. Adaptation of the main μKOS makefile

Adaptation of the file /Target/Baphomet_746/Variant_Test/System/makefile (i.e. for the **CSEM_VIP_777** target).

```
...

SHELL           = /bin/sh

# uKOS System description
# ----

# Project paths

# - PATH_UKOS --> Main uKOS folder
# - PATH_PORT --> Main uKOS port folders
# - PATH_BASE --> MyProjects Root Base folders
# - PATH_VARI --> MyProjects Root Variant_xx folders
# - PATH_MAPP --> Linker script folder
# - PATH_USER --> User project folder

PATH_UKOS        = $(PATH_UKOS_KERNEL)
PATH_PORT        = $(PATH_UKOS_KERNEL)/Ports
PATH_BASE        = $(PATH_UKOS_KERNEL)/Ports/Targets/$(BOARD)/Base
PATH_VARI        = $(PATH_UKOS_KERNEL)/Ports/Targets/$(BOARD)/$(V)
PATH_MAPP        = $(PATH_BASE)/Runtime

# Target & Infrastructure

# - VERSIONING          --> SVN
# - VERSIONING          --> GIT
# - VERSIONING          --> NONE

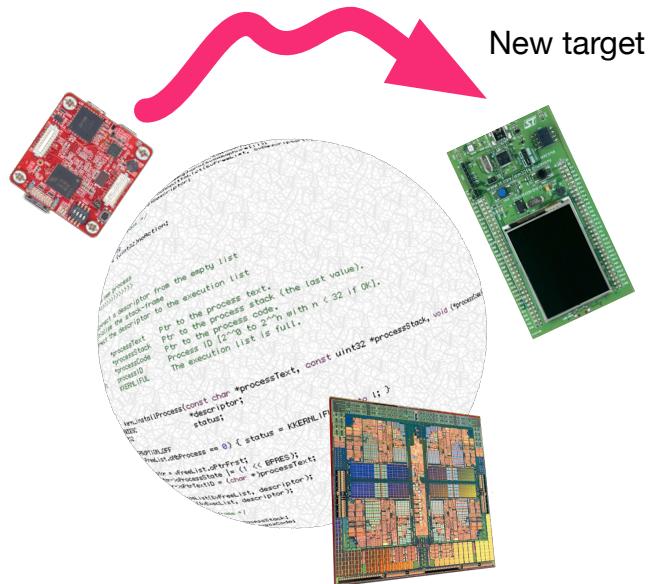
PREFIX           = arm-none-eabi-
BOARD            = CSEM_VIP_777
V                = Variant_Test
SOC               = STM32F777
CORE              = CORTEX_M7
STANDARD         = -std=c99
ISO_IEC_9899     = -pedantic -ansi
OPTIMISATION     = -Os
VERSIONING       = SVN

FLAGS_USER        = -D__ROMABLE__
```

```
...  
TOOL           = $(PATH_UKOS)/Tools/bangla/BaNgLa.o  
TOOL           += $(PATH_UKOS)/Tools/binfill/binfill.o  
TOOL           += $(PATH_UKOS)/Tools/console/console.o  
...  
TOOL           += $(PATH_UKOS)/Tools/viewer/viewer.o  
TOOL           += $(PATH_APPLS)/uKOS_Apps/sqe_kalman.o  
TOOL_HIDE      =  
  
# The protocol components  
  
...
```

D. Annex

Porting μKOS



D.1. Introduction

Porting μKOS to another target is not a complex task. First of all, it is necessary to have a good overview of the system organization (please, read all the μKOS book). This document, by a concrete example, will explain step-by-step all the operations necessary for porting and for configuring μKOS for a new target. To simplify the explanation it is considered only the porting on targets having an already supported CPU. Of course, other CPUs can be used, but in this case, the porting is much more complex.

D.2. Considered target: ST Discovery 429 board

In the considered example it is proposed to port μKOS on the ST Discovery-429 target. The used reference target is the Discovery-407. These two targets have been upgraded with additional peripherals in order to extend their capabilities. The upgraded configurations consist in:

- JTAG via the FT4232 chip.
- 2 additional UARTs (3 for the Discovery 429) via the FT4232 chip.
- 1 EEPROM (for storing μKOS applications) & interface of the SDRAM, (only for the Discovery 429).
- 1 μSDCard interface.
- 8 additional LEDs & 1 rotary switch (4-bits).

Function	μKOS		Peripheral	
	Manager		Discovery 407	Discovery 429
JTAG		-	JTAG	JTAG
Serial KURT0		urt0	USART2	UART4
Serial KURT1		urt1	USART2	UART5
Serial KURT2		urt2	-	USART1
μKOS EEPROM		glob	SPI2, PORTS: B12	SPI4, PORTS: E4
μSDCard		uscd	SPI3, PORTS: C9, E2	SPI4, PORTS: C13, A5
8 LEDs		misc	PORTS: C1, C2, E6, C13, D7, D6, D10, D11	PORTS: E3, B7, B4, D5, D4, C8, F6, C3
Rotary switch		misc	PORTS: D0, D1, D3, D4	PORTS: G3, G2
SDRAM		-	-	FMC

D.2.1. Preparing the software package

As indicated, the new Discovery-429 target is similar to the Discovery-407 one. So, the first step should be to generate a project structure for this new target. This step could be easily achieved by copy the Discovery-407 project in the Discovery-429.

```
cd ${PATH_UKOS_KERNEL}/Ports/Targets  
cp -r Discovery_407 Discovery_429
```

D.2.2. Modify the necessary files

Now the customization process starts. Step-by-step it is necessary to modify the target specific files. The colour indicates the degree of complexity for such a modification. Anyway, it is strongly suggested to visualize and to compare the marked files; this will help to understand the porting.

Low complexity modifications

Moderate complexity modifications

Serious complexity modifications

Need to be rewritten / or deleted

./README

./Variant_Test/Includes/Board/board.h
./Variant_Test/Includes/System/conf_kernel.h
./Variant_Test/Includes/System/conf_system.h

./Variant_Test/Runtime/init.c
./Base/Runtime/cmns.c
./Base/Runtime/exce.c
./Base/Runtime/link.ld
./Base/Runtime/link_App.ld

./Variant_Test/Lib_gene/usdc/stub_usdc_sdcard_spi3.c
./Variant_Test/Lib_tbox/misc/stub_misc.c
./Base/Lib_tbox/glob/stub_glob.c

./Variant_Test/Processes/start_up/stub_start_up.c
./Base/Processes/alive/stub_alive.c

./Variant_Test/System/jtag_FTDI.cfg
./Variant_Test/System/jtag_Stlink.cfg
./Variant_Test/System/makefile

./Base/Lib_comm/comm/comm.c
./Base/Lib_comm/urt0/stub_urt0_usart2.c
./Base/Lib_comm/urt1/stub_urt1_usart3.c >> ./Base/Lib_comm/urt1/stub_urt1_usart1.c
./Base/Lib_kern/kern/stub_kern_tim_x_y_z_yyy.c
./Base/Lib_peri/adc0/stub_adc0.c

./Variant_Test/Shared/spi4/spi4.h
./Variant_Test/Shared/spi4/spi4.c

D.2.3. Building the new system

Now that all the modification are in place, we can create a script for building the complete project.

```
cd ${PATH_UKOS_KERNEL}/Ports/Targets/Discovery_429/Variant_Test/System  
make
```

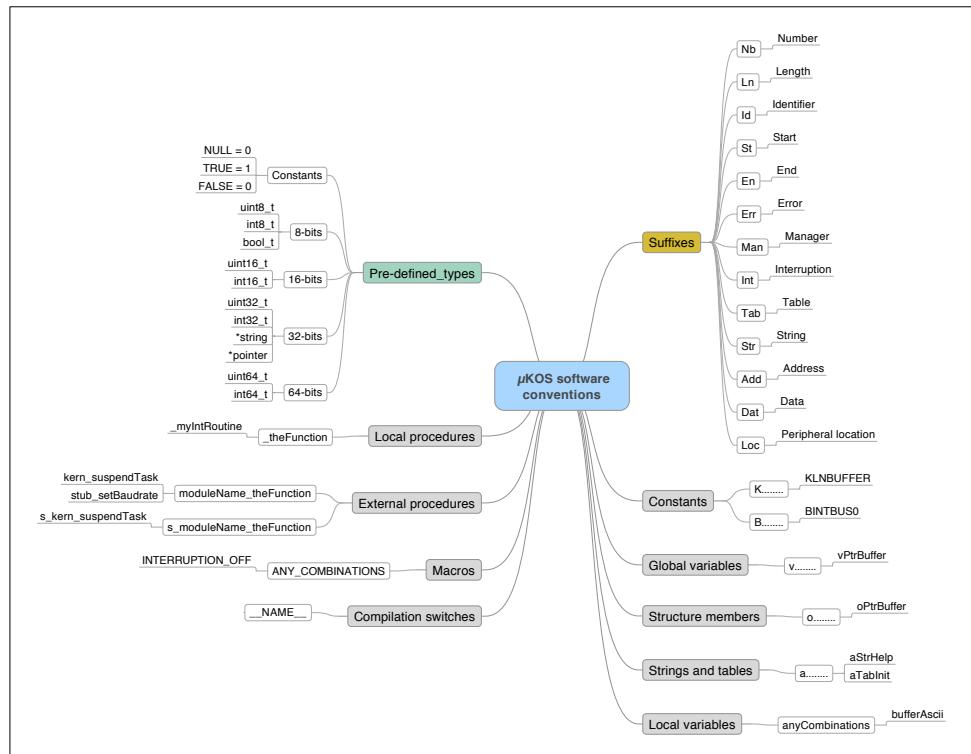

E. Annex

The μKOS package



E.1. Rules and conventions used for the project

In order to have and to maintain a coherency along all the μKOS project, the following rules for writing software have been used.



The used text editor is the **BBEDIT**; the ASCII files (*.c, *.h, *.s, and *.ld) are formatted in the following way:

Font: **Monaco 9-points**

Tab: **4 spaces**

Line termination: **LF (UNIX)**

E.2. The package

The downloaded µKOS package contains all the schematics, sources, makefiles and tools necessary for building the system and for creating applications. Before starting, it is necessary to create the `.bash_profile` (for OSX) or `.bash_aliases` (for Ubuntu). Then, additional packages need to be downloaded and installed.

E.2.1. Prerequisites for OSX (10.12.x)

Configure the .bash_profile

```
-----  
# Author:      Franzi Edo.  The 2011-01-05  
# Modifs:  
#  
# SVN:  
# $Author:: efr           $: Author of last commit  
# $Rev:: 46              $: Revision of last commit  
# $Date:: 2016-05-16 21:57:26#$: Date of last commit  
#  
# Project:     uKOS  
# Goal:       Main profile.  
#  
# (c) 1992-2017, Franzi Edo.  
# -----  
#  
# Franzi Edo.          _ _ / / / \ \ / _ /  
# 5-Route de Cheseaux   / / / , < / / / \ \ _ \ /  
# CH 1400 Cheseaux-Noréaz / / / / | / / / / _ / /  
#                           \_,/_/ |_\_ / / / _ / /  
# edo.franzi@ukos.ch  
#  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU Affero General Public License as published by  
# the Free Software Foundation, either version 3 of the License, or  
# (at your option) any later version.  
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU Affero General Public License for more details.  
#  
# You should have received a copy of the GNU Affero General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>.  
#-----
```

```
# Set the PATH for efr development environment
# -----
#
PATH=${PATH}:/opt/local/bin
PATH=${PATH}:/opt/local/sbin
PATH=${PATH}:/opt/subversion/bin
PATH=${PATH}:/usr/local/bin
PATH=${PATH}:/usr/bin

MANPATH=${MANPATH}:/opt/local/share/man

# Project environment
# -----
#
# PATH_PORTS_ROOT = Macports tools
# PATH_UKOS_ROOT = uKOS Project (active workspace)
# PATH_UKOS_KERNEL = uKOS kernel

PATH_PORTS_ROOT=/opt/local
PATH_UKOS_ROOT=${HOME}/DATA/DATA_Private/uKOS_Project/uKOS_Soft
PATH_UKOS_KERNEL=${PATH_UKOS_ROOT}/OS_Kernel-III

# Cross compilation environment
# -----
#
# PATH_TOOLS_ROOT      = Tool location
# PATH_TOOLS_GCC        = GCC root location
# PATH_TOOLS_UNIX       = My BIN UNIX tool location
# PATH_FTDI_INCLUDES   = FTDI includes
# PATH_FTDI_LIBRARIES = FTDI libraries

PATH_TOOLS_ROOT=/opt/uKOS
PATH_TOOLS_GCC=${PATH_TOOLS_ROOT}
PATH_TOOLS_UNIX=${PATH_TOOLS_ROOT}/bin
PATH_FTDI_INCLUDES=/usr/local/include
PATH_FTDI_LIBRARIES=/usr/local/lib

PATH_GCC_ICYFLEX=${PATH_TOOLS_GCC}/cross/gcc-4.6.4/icyflex1
PATH_GCC_COLFIRE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/coldfire
PATH_GCC_MC6833X=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/cpu32
PATH_GCC_BLACKFIN=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/blackfin
PATH_GCC_CORTEX3=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/arm
PATH_GCC_CORTEX7=${PATH_TOOLS_GCC}/cross/gcc-M7-5.4.1/arm
PATH_GCC_CNATIVE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/osx
PATH_GCC_GPUTILS=${PATH_TOOLS_GCC}/cross/gputils-1.5.0-1
PATH_GCC_SDCCPIC=${PATH_TOOLS_GCC}/cross/sdcc-3.6.0
```

```
PATH=${PATH}: ${PATH_TOOLS_UNIX}
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/openocd-0.10.0/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/doxygen-1.8.13/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/graphviz-2.40.1/bin
PATH=${PATH}: ${PATH_GCC_ICYFLEX}/bin
PATH=${PATH}: ${PATH_GCC_COLFIRE}/bin
PATH=${PATH}: ${PATH_GCC_MC6833X}/bin
PATH=${PATH}: ${PATH_GCC_BLAKFIN}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX3}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX7}/bin
PATH=${PATH}: ${PATH_GCC_CNATIVE}/bin
PATH=${PATH}: ${PATH_GCC_GPUTILS}/bin
PATH=${PATH}: ${PATH_GCC_SDCCPIC}/bin

export PATH_FTDI_INCLUDES
export PATH_FTDI_LIBRARIES
export PATH_TOOLS_GCC
export PATH_TOOLS_UNIX
export PATH_UKOS_KERNEL
export PATH_GCC_ICYFLEX
export PATH_GCC_COLFIRE
export PATH_GCC_MC6833X
export PATH_GCC_BLAKFIN
export PATH_GCC_CORTEX3
export PATH_GCC_CORTEX7
export PATH_GCC_CNATIVE
export PATH_GCC_GPUTILS
export PATH_GCC_SDCCPIC

export MANPATH
export PATH
```

Install the Macports and some additional packages

Follow the instructions: <http://www.macports.org>

```
xcode-select --install

sudo port install apple-gcc42
sudo port install automake
sudo port install cmake
sudo port install libftdil
sudo port install libtool
sudo port install autoconf
sudo port install pkgconfig
sudo port install coreutils
sudo port install wget
```

E.2.2. Prerequisites for Ubuntu (16.04 LTS)

Configure the .bash_aliases

```
-----  
# Author:      Franzi Edo.  The 2011-01-05  
# Modifs:  
#  
# SVN:  
# $Author:: efr          $: Author of last commit  
# $Rev:: 46            $: Revision of last commit  
# $Date:: 2016-05-16 21:57:26#$: Date of last commit  
#  
# Project:     uKOS  
# Goal:        Main profile.  
#  
# (c) 1992-2017, Franzi Edo.  
# -----  
#  
# Franzi Edo.           _ _ / / / \ \ / / / /  
# 5-Route de Cheseaux    / / / , < / / / / \ \ \ /  
# CH 1400 Cheseaux-Noréaz / / / / | / / / / / /  
#                           \_,/_/ |_\_\_//__/  
# edo.franzi@ukos.ch  
#  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU Affero General Public License as published by  
# the Free Software Foundation, either version 3 of the License, or  
# (at your option) any later version.  
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU Affero General Public License for more details.  
#  
# You should have received a copy of the GNU Affero General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>.  
#-----  
  
# Set the PATH for efr development environment  
# -----  
  
PATH=${PATH}:/opt/local/bin  
PATH=${PATH}:/opt/local/sbin  
PATH=${PATH}:/opt/subversion/bin  
PATH=${PATH}:/usr/local/bin  
PATH=${PATH}:/usr/bin
```

```
MANPATH=${MANPATH}:/opt/local/share/man

# Project environment
# -----
#
# # PATH_UKOS_ROOT      = uKOS Project (active workspace)
# # PATH_UKOS_KERNEL    = uKOS kernel

PATH_UKOS_ROOT=${HOME}/DATA/DATA_Private/uKOS_Project/uKOS_Soft
PATH_UKOS_KERNEL=${PATH_UKOS_ROOT}/OS_Kernel-III

# Cross compilation environment
# -----
#
# # PATH_TOOLS_ROOT      = Tool location
# # PATH_TOOLS_GCC        = GCC root location
# # PATH_TOOLS_UNIX       = My BIN UNIX tool location
# # PATH_FTDI_INCLUDES   = FTDI includes
# # PATH_FTDI_LIBRARIES  = FTDI libraries

PATH_TOOLS_ROOT=/opt/uKOS
PATH_TOOLS_GCC=${PATH_TOOLS_ROOT}
PATH_TOOLS_UNIX=${PATH_TOOLS_ROOT}/bin
PATH_FTDI_INCLUDES=${PATH_TOOLS_GCC}/Packages/libftd2xx-1.4.6
PATH_FTDI_LIBRARIES=${PATH_TOOLS_GCC}/Packages/libftd2xx-1.4.6/build

PATH_GCC_ICYFLEX=${PATH_TOOLS_GCC}/cross/gcc-4.6.4/icyflex1
PATH_GCC_COLFIRE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/coldfire
PATH_GCC_MC6833X=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/cpu32
PATH_GCC_BLACKFIN=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/blackfin
PATH_GCC_CORTEX3=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/arm
PATH_GCC_CORTEX7=${PATH_TOOLS_GCC}/cross/gcc-M7-5.4.1/arm
PATH_GCC_GPUTILS=${PATH_TOOLS_GCC}/cross/gputils-1.5.0-1
PATH_GCC_SDCCPIC=${PATH_TOOLS_GCC}/cross/sdcc-3.6.0
```

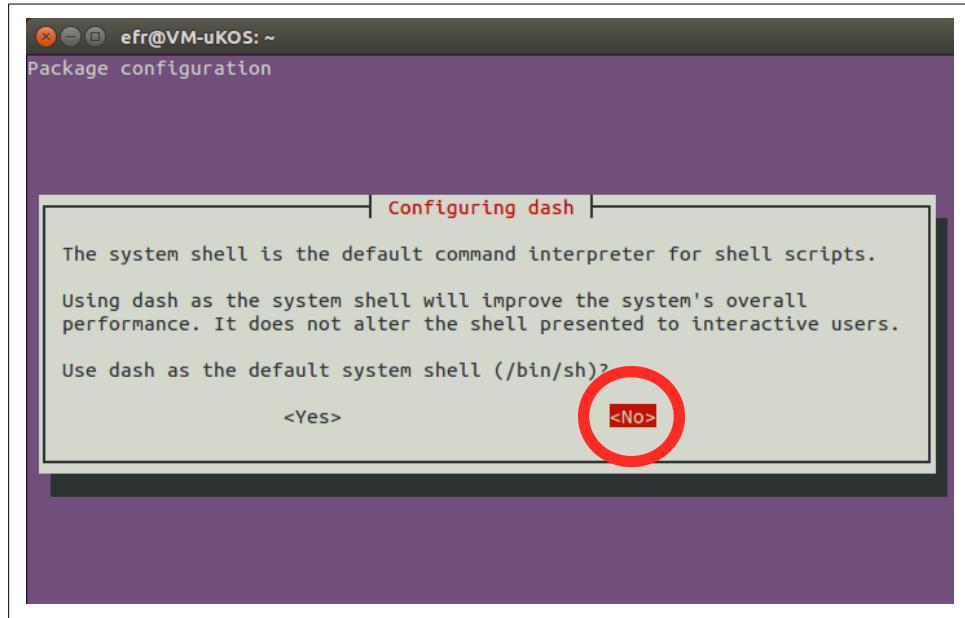
```
PATH=${PATH}: ${PATH_TOOLS_UNIX}
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/openocd-0.10.0/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/doxygen-1.8.13/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/graphviz-2.40.1/bin
PATH=${PATH}: ${PATH_GCC_ICYFLEX}/bin
PATH=${PATH}: ${PATH_GCC_COLFIRE}/bin
PATH=${PATH}: ${PATH_GCC_MC6833X}/bin
PATH=${PATH}: ${PATH_GCC_BLAKFIN}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX3}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX7}/bin
PATH=${PATH}: ${PATH_GCC_GPUTILS}/bin
PATH=${PATH}: ${PATH_GCC_SDCCPIC}/bin

export PATH_FTDI_INCLUDES
export PATH_FTDI_LIBRARIES
export PATH_TOOLS_GCC
export PATH_TOOLS_UNIX
export PATH_UKOS_KERNEL
export PATH_GCC_ICYFLEX
export PATH_GCC_COLFIRE
export PATH_GCC_MC6833X
export PATH_GCC_BLAKFIN
export PATH_GCC_CORTEX3
export PATH_GCC_CORTEX7
export PATH_GCC_GPUTILS
export PATH_GCC_SDCCPIC

export MANPATH
export PATH
```

Activate the bash

```
sudo dpkg-reconfigure dash
```



Install some additional packages

```
sudo apt-get install build-essential  
sudo apt-get install libtool bison flex gawk m4 texinfo automake  
sudo apt-get install libncurses-dev libusb-1.0-0-dev libusb-dev zlib1g-dev  
sudo apt-get install cmake  
sudo apt-get install gtkterm  
sudo apt-get install libpcre3 libpcre3-dev  
sudo apt-get install libgl1-mesa-dev  
sudo apt install git  
sudo apt-get install libvte-dev intltool build-essential libgtk2.0-dev  
sudo apt-get install ttf-mscorefonts-installer culmus  
  
sudo usermod -a -G dialout ukos
```

Add a new rule for using the FTDI chips

Create a file 90-ukos-ftdi.rules containing these rules:

```
ACTION!="add|change", GOTO="ukos_ftdi_rules_end"
SUBSYSTEM!="usb|tty", GOTO="ukos_ftdi_rules_end"

ATTRS{idProduct}=="6011", ATTRS{idVendor}=="0403", ATTRS{product}=="uKOS - Kernel",
GROUP="dialout"
ATTRS{idProduct}=="6011", ATTRS{idVendor}=="0403", ATTRS{product}=="uKOS - Kernel",
ATTR{bInterfaceNumber}=="00", RUN+="/bin/sh -c 'echo $kernel > /sys/bus/usb/drivers/
ftdi_sio/unbind'

ATTRS{idProduct}=="6015", ATTRS{idVendor}=="0403", ATTRS{product}=="uKOS - FIFO",
GROUP="dialout"
ATTRS{idProduct}=="6015", ATTRS{idVendor}=="0403", ATTRS{product}=="uKOS - FIFO",
RUN+="/bin/sh -c 'echo $kernel > /sys/bus/usb/drivers/ftdi_sio/unbind'"

LABEL="ukos_ftdi_rules_end"
```

Copy the file into the rules.d folder

```
cd etc/udev/rules.d
sudo cp xyz/90-ukos-ftdi.rules .
```

E.2.3. Download & install the necessary open source packages

Put at the same level all the packages: **gcc-4.6.4**, **gcc-7.2.0**, **gdb-8.0.0**, **binutils-2.29**, **newlib-2.5.0** ...

In my case I created **\$[PATH_TOOLS_GCC]/Packages** and I put all the downloaded packages.

Necessary packages used for building 32-bit uKOS kernel/applications

```
wget ftp://sourceware.org/pub/binutils/releases/binutils-2.29.tar.bz2
wget ftp://ftp.gnu.org/pub/gnu/gcc/gcc-7.2.0/gcc-7.2.0.tar.bz2
wget ftp://ftp.gnu.org/pub/gnu/gdb/gdb-8.0.0.tar.gz
wget ftp://sourceware.org/pub/newlib/newlib-2.5.0.tar.gz
wget http://www.ftdichip.com/Drivers/D2XX/libftd2xx-x86_64-1.4.6.tgz
wget http://www.ftdichip.com/Drivers/D2XX/MacOSX/D2XX1.4.4.dmg
wget http://www.ivarch.com/programs/sources/pv-1.6.0.tar.bz2
wget http://prdownloads.sourceforge.net/swig/swig-3.0.12.tar.gz
git clone git://git.code.sf.net/p/openocd/code openocd-0.10.0
https://developer.arm.com/-/media/Files/downloads/gnu-rm/6_1-2017q1/
    gcc-arm-none-eabi-6-2017-q1-update-mac.tar.bz2?
    product=GNU%20ARM%20Embedded%20Toolchain,64-bit,,Mac%20OS%20X,6-2017-q1-update
https://developer.arm.com/-/media/Files/downloads/gnu-rm/6_1-2017q1/
    gcc-arm-none-eabi-6-2017-q1-update-linux.tar.bz2?
    product=GNU%20ARM%20Embedded%20Toolchain,64-bit,,Linux,6-2017-q1-update
```

Necessary packages used for building 8-bit uKOS applications

```
wget http://sourceforge.net/projects/boost/files/boost/1.64.0/boost_1_64_0.tar.bz2
wget https://sourceforge.net/projects/gptools/files/latest/download?source=files
wget https://github.com/psmay/pk2cmd/archive/master.zip
wget http://sourceforge.net/projects/sdcc/files/sdcc/3.6.0/sdcc-src-3.6.0.tar.bz2
```

Necessary packages for documenting uKOS

```
wget ftp://ftp.stack.nl/pub/users/dimitri/doxygen-1.8.13.src.tar.gz
wget http://www.graphviz.org/pub/graphviz/stable/SOURCES/graphviz-2.40.1.tar.gz
```

E.2.4. Install the “cross-compilers”

Just follow the indications available in the Annex E “**GCC Toolchains**”.

E.2.5. Build additional μKOS UNIX tools

Follow these steps:

1. Launch a terminal.
2. Enter into the folder: **cd \${PATH_UKOS_KERNEL}/Ports/Tools/UNIX_Tools.**
3. Cleanup all the binaries: **sh clean.sh.**
4. Build the system: **sh build.sh.**

E.2.6. Build the full μKOS system

Now that everything should be in place, the μKOS system could be created. Select the target that you want to build (i.e. Baphomet_407, Discovery_429, etc.) and follow these steps (i.e. for the **Baphomet_746** target):

1. Launch a terminal.
2. Enter into the folder: ...
... **cd \${PATH_UKOS_KERNEL}/Ports/Targets/Baphomet_746/Variant_1/System.**
3. Build the system: **make.**

If everything has been configured as explained, at the end of the “**make**” the μKOS system is available to be downloaded and re-flashed **\${PATH_UKOS_KERNEL}/Ports/Targets/Baphomet_746/Variant_1/System/EPROM.elf**.

E.2.7. Build the standalone μKernel application

A standalone application running without the μKOS OS is available. Follow these steps:

1. Launch a terminal.
2. Enter into the folder: ...
... **cd \${PATH_UKOS_KERNEL}/Applications/uKOS_Apps/cmpl_std/Baphomet_746.**
3. Build the application: **make.**

E.2.8. Build all the example applications

Follow these steps:

1. Launch a terminal.
2. Enter into the folder: ...
... **cd \${PATH_UKOS_KERNEL}/Applications/uKOS_Apps.**
3. Cleanup all the binaries: **sh clean.sh.**
4. Build the system: **sh build.sh.**

E.2.9. Build the “doxygen” documentation

1. Launch a terminal.
2. Build the documentation **doxygen \${PATH_UKOS_KERNEL}/Doxygen_Doc/tmp/Doxyfile.dox.**

E.3. Setting-up the Eclipse environment

Eclipse can be easily configured for managing all the uKOS project. All the following points show the minimal configuration steps to be operational. The following configuration does not allow Eclipse to manage the project makefiles. Eclipse is used just for editing and for launching the conventional makefiles for building the programs.

E.3.1. Download the Eclipse package

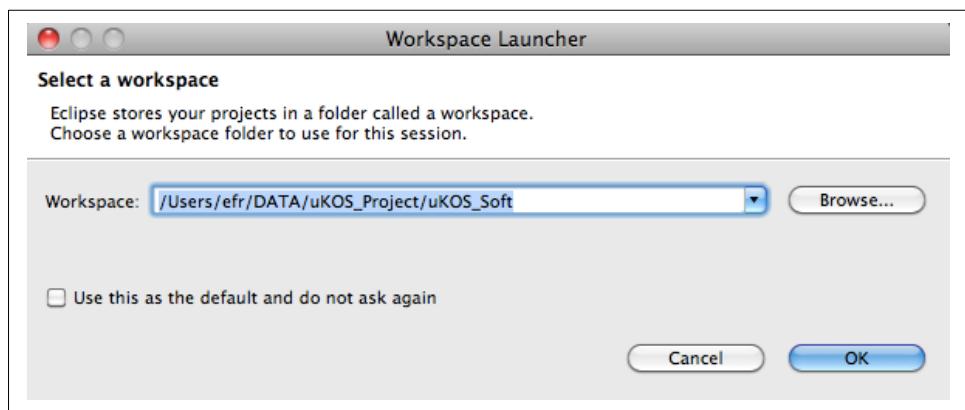
Eclipse Mars: <http://www.eclipse.org>

Before launching Eclipse, it is necessary to know where the project and the gcc compilers are located. As an example, here are the locations I use on my machine:

```
PATH_TOOLS_ROOT= Location of the development packages  
PATH_TOOLS_GCC=${PATH_TOOLS_ROOT}  
PATH_TOOLS_UNIX=${PATH_TOOLS_ROOT}/bin  
PATH_UKOS_ROOT=${HOME}DATA/uKOS_Project/uKOS_Soft/OS_Kernel-III  
  
PATH_GCC_ICYFLEX=${PATH_TOOLS_GCC}/cross/gcc-4.6.4/icyflex1  
PATH_GCC_COLFIRE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/coldfire  
PATH_GCC_MC6833X=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/cpu32  
PATH_GCC_BLACKFIN=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/blackfin  
PATH_GCC_CORTEX3=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/arm  
PATH_GCC_CORTEX7=${PATH_TOOLS_GCC}/cross/gcc-M7-5.4.1/arm
```

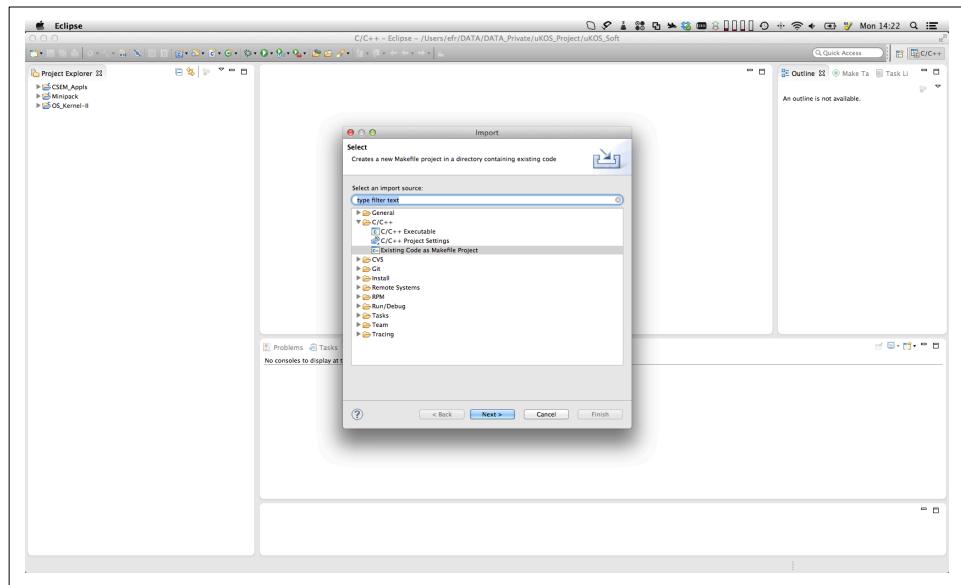
E.3.2. Configuring Eclipse

The first time that Eclipse is launched, it asks for the path of the workspace. For this example, the workspace has to be set at this location: \${HOME}DATA/uKOS_Project/uKOS_Soft.



Importing the project into the Eclipse workspace

1. From the menu **File** select **Import**.
2. In the window **Select**, choose **C/C++ -> Existing code as Makefile Project**.
3. **Next**.
4. **Browse** to locate the project that has to be imported ...
... \${HOME}/DATA/uKOS_Project/uKOS_Soft/OS_Kernel-III.



Now the project is imported in the Eclipse workspace. At this stage it is necessary to set-up the project paths (for the gcc compiler and for the tools), as well as for the c project sources.

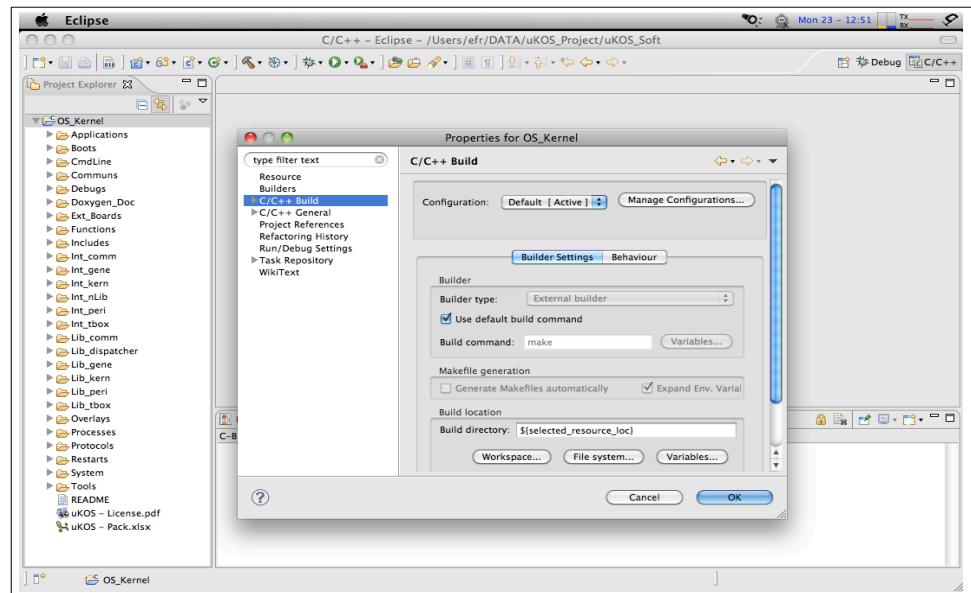
- From the menu **Eclipse -> Preferences -> C/C++ -> Build** select **Environment**.
- Add all the necessary **Variables** and their **Values**.

Variable	Value
PATH	/usr/bin:/usr/local/bin
PATH_GCC_CORTEX3	/opt/uKOS/cross/gcc-7.2.0/arm
PATH_GCC_CORTEX7	/opt/uKOS/cross/gcc-M7-5.4.1/arm
PATH_GCC_ICYFLEX	/opt/uKOS/cross/gcc-4.6.4/icyflex1
PATH_TOOLS_UNIX	/opt/uKOS/bin
PATH_UKOS_KERNEL	/Users/efr/DATA/uKOS_Project/uKOS_Soft/OS_Kernel-III

In the menu **Project** be sure that the item **Build automatically** is not selected!

Preparing for supporting the makefiles in the building process.

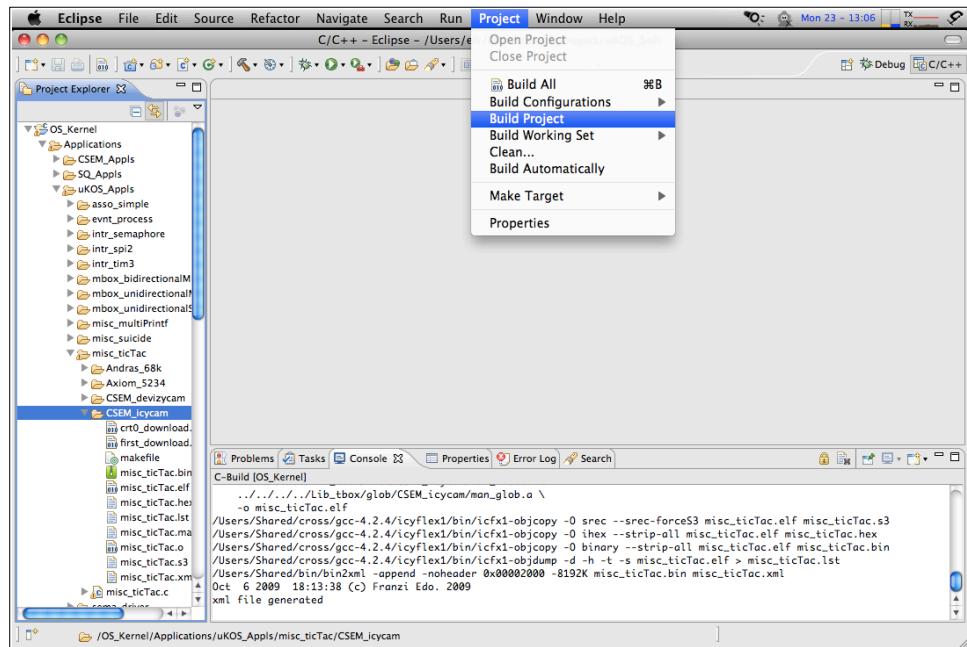
- From the menu **Project -> Properties** select **C/C++ Building**.



Now Eclipse is configured and ready to operate with the μKOS project.

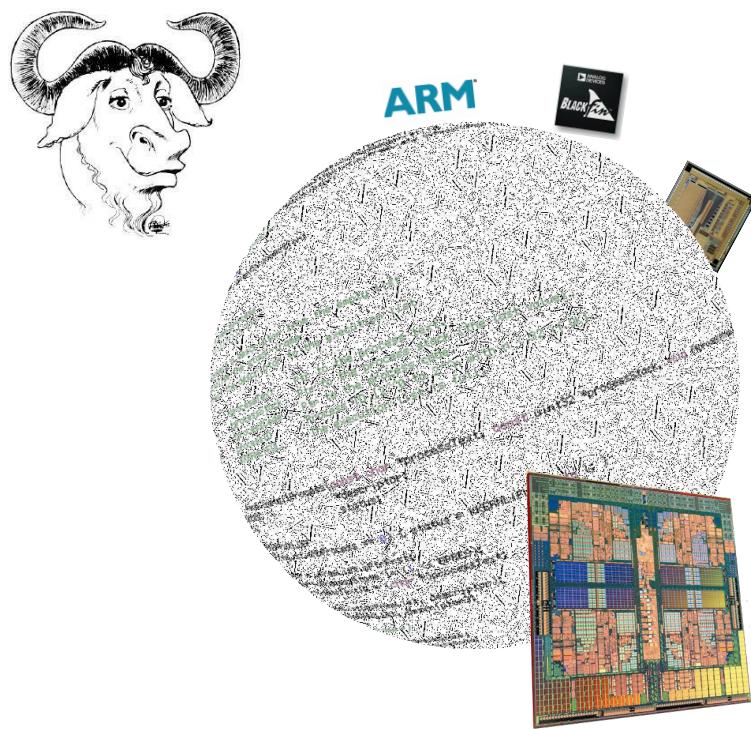
For compiling a program just proceed in this way:

1. In the menu **Run -> External Tools** select the target on which you want to operate; i.e. **Set target = icycam**.
2. On the left there is the tree of the μKOS project. Select the project target you want to build (i.e. the **misc_ticTac**), then build the project (in the menu **Project -> Build Project**). This launches the makefile that corresponds to the selected project.



F. Annex

gcc toolchains (m68k, ColdFire, Blackfin, icyflex-1, ARM Cortex Mx)



F.1. The gcc compiler

OSX is now a UNIX machine, so the usage of the gcc tools is possible. Here are the necessary steps for building cpu32, ColdFire, Blackfin, Cortex and IcyCAM cross-compilers running in OSX hosts. On **Ubuntu**, just proceed in the same way.

μKOS is based on a μKernel allowing to run many programs at the same time. This means that even the **newlib** should be configured to support reentrant system calls.

F.1.1. Download the necessary packages

Necessary packages used for building 32-bit uKOS kernel/applications

```
wget ftp://sourceware.org/pub/binutils/releases/binutils-2.29.tar.bz2
wget ftp://ftp.gnu.org/pub/gnu/gcc/gcc-7.2.0/gcc-7.2.0.tar.bz2
wget ftp://ftp.gnu.org/pub/gdb/gdb-8.0.0.tar.gz
wget ftp://sourceware.org/pub/newlib/newlib-2.5.0.tar.gz
wget http://www.ftdichip.com/Drivers/D2XX/libftd2xx-x86_64-1.4.6.tgz
wget http://www.ftdichip.com/Drivers/D2XX/MacOSX/D2XX1.4.4.dmg
wget http://www.ivarch.com/programs/sources/pv-1.6.0.tar.bz2
wget http://prdownloads.sourceforge.net/swig/swig-3.0.12.tar.gz
git clone git://git.code.sf.net/p/openocd/code openocd-0.10.0
https://developer.arm.com/-/media/Files/downloads/gnu-rm/6_1-2017q1/
    gcc-arm-none-eabi-6-2017-q1-update-mac.tar.bz2?
    product=GNU%20ARM%20Embedded%20Toolchain,64-bit,,Mac%20OS%20X,6-2017-q1-update
https://developer.arm.com/-/media/Files/downloads/gnu-rm/6_1-2017q1/
    gcc-arm-none-eabi-6-2017-q1-update-linux.tar.bz2?
    product=GNU%20ARM%20Embedded%20Toolchain,64-bit,,Linux,6-2017-q1-update
```

Necessary packages used for building 8-bit uKOS applications

```
wget http://sourceforge.net/projects/boost/files/boost/1.64.0/boost_1_64_0.tar.bz2
wget https://sourceforge.net/projects/gputils/files/latest/download?source=files
wget https://github.com/psmay/pk2cmd/archive/master.zip
wget http://sourceforge.net/projects/sdcc/files/sdcc/3.6.0/sdcc-src-3.6.0.tar.bz2
```

Necessary packages for documenting uKOS

```
wget ftp://ftp.stack.nl/pub/users/dimitri/doxygen-1.8.13.src.tar.gz
wget http://www.graphviz.org/pub/graphviz/stable/SOURCES/graphviz-2.40.1.tar.gz
```

F.1.2. Prepare the environment

Edit the file **.bash_profile** for OSX or the **.bash_aliases** for Ubuntu in your home directory and add the following lines; (of course these paths have to be coherent with the next installation steps).

Configure the .bash_profile

```
-----  
# Author:      Franzi Edo.  The 2011-01-05  
# Modifs:  
#  
# SVN:  
# $Author::: efr           $: Author of last commit  
# $Rev::: 46             $: Revision of last commit  
# $Date::: 2016-05-16 21:57:26#$: Date of last commit  
#  
# Project:    uKOS  
# Goal:       Main profile.  
#  
# (c) 1992-2017, Franzi Edo.  
# -----  
#  
# Franzi Edo.          _ _/_/_/\_ \_/_/  
# 5-Route de Cheseaux   / / / ,< / / / /\_ \_  
# CH 1400 Cheseaux-Noréaz / / / / | / _/ / _/ /  
#                           \_,/_/ |_\_/_//__/_/  
# edo.franzi@ukos.ch  
#  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU Affero General Public License as published by  
# the Free Software Foundation, either version 3 of the License, or  
# (at your option) any later version.  
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU Affero General Public License for more details.  
#  
# You should have received a copy of the GNU Affero General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>.  
#-----  
  
# Set the PATH for efr development environment  
# -----  
  
PATH=${PATH}:/opt/local/bin  
PATH=${PATH}:/opt/local/sbin  
PATH=${PATH}:/opt/subversion/bin  
PATH=${PATH}:/usr/local/bin  
PATH=${PATH}:/usr/bin  
  
MANPATH=${MANPATH}:/opt/local/share/man
```

```
# Project environment
# -----
#
# PATH_PORTS_ROOT = Macports tools
# PATH_UKOS_ROOT = uKOS Project (active workspace)
# PATH_UKOS_KERNEL = uKOS kernel

PATH_PORTS_ROOT=/opt/local
PATH_UKOS_ROOT=${HOME}/DATA/DATA_Private/uKOS_Project/uKOS_Soft
PATH_UKOS_KERNEL=${PATH_UKOS_ROOT}/OS_Kernel-III

# Cross compilation environment
# -----
#
# PATH_TOOLS_ROOT = Tool location
# PATH_TOOLS_GCC = GCC root location
# PATH_TOOLS_UNIX = My BIN unix tool location
# PATH_FTDI_INCLUDES = FTDI includes
# PATH_FTDI_LIBRARIES = FTDI libraries

PATH_TOOLS_ROOT=/opt/uKOS
PATH_TOOLS_GCC=${PATH_TOOLS_ROOT}
PATH_TOOLS_UNIX=${PATH_TOOLS_ROOT}/bin
PATH_FTDI_INCLUDES=/usr/local/include
PATH_FTDI_LIBRARIES=/usr/local/lib

PATH_GCC_ICYFLEX=${PATH_TOOLS_GCC}/cross/gcc-4.6.4/icyflex1
PATH_GCC_COLDFIRE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/coldfire
PATH_GCC_MC6833X=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/cpu32
PATH_GCC_BLACKFIN=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/blackfin
PATH_GCC_CORTEX3=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/arm
PATH_GCC_CORTEX7=${PATH_TOOLS_GCC}/cross/gcc-M7-5.4.1/arm
PATH_GCC_CNATIVE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/osx
PATH_GCC_GPUTILS=${PATH_TOOLS_GCC}/cross/gputils-1.5.0-1
PATH_GCC_SDCCPIC=${PATH_TOOLS_GCC}/cross/sdcc-3.6.0
```

```
PATH=${PATH}: ${PATH_TOOLS_UNIX}
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/openocd-0.10.0/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/doxygen-1.8.13/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/graphviz-2.40.1/bin
PATH=${PATH}: ${PATH_GCC_ICYFLEX}/bin
PATH=${PATH}: ${PATH_GCC_COLFIRE}/bin
PATH=${PATH}: ${PATH_GCC_MC6833X}/bin
PATH=${PATH}: ${PATH_GCC_BLAKFIR}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX3}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX7}/bin
PATH=${PATH}: ${PATH_GCC_CNATIVE}/bin
PATH=${PATH}: ${PATH_GCC_GPUTILS}/bin
PATH=${PATH}: ${PATH_GCC_SDCCPIC}/bin

export PATH_FTDI_INCLUDES
export PATH_FTDI_LIBRARIES
export PATH_TOOLS_GCC
export PATH_TOOLS_UNIX
export PATH_UKOS_KERNEL
export PATH_GCC_ICYFLEX
export PATH_GCC_COLFIRE
export PATH_GCC_MC6833X
export PATH_GCC_BLAKFIR
export PATH_GCC_CORTEX3
export PATH_GCC_CORTEX7
export PATH_GCC_CNATIVE
export PATH_GCC_GPUTILS
export PATH_GCC_SDCCPIC

export MANPATH
export PATH
```

Configure the .bash_aliases

```
-----  
# Author:      Franzi Edo.  The 2011-01-05  
# Modifs:  
#  
# SVN:  
# $Author::: efr           $: Author of last commit  
# $Rev::: 46             $: Revision of last commit  
# $Date::: 2016-05-16 21:57:26#$: Date of last commit  
#  
# Project:    uKOS  
# Goal:       Main profile.  
#  
# (c) 1992-2017, Franzi Edo.  
# -----  
#  
# Franzi Edo.          _ _/_/_/\_ \_/_/  
# 5-Route de Cheseaux   / / / ,< / / / /\_ \  
# CH 1400 Cheseaux-Noréaz / / / / | / _/ / _/ /  
#                           \_,/_/ |_\_/_//__/  
# edo.franzi@ukos.ch  
#  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU Affero General Public License as published by  
# the Free Software Foundation, either version 3 of the License, or  
# (at your option) any later version.  
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU Affero General Public License for more details.  
#  
# You should have received a copy of the GNU Affero General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>.  
#-----  
  
# Set the PATH for efr development environment  
# -----  
  
PATH=${PATH}:/opt/local/bin  
PATH=${PATH}:/opt/local/sbin  
PATH=${PATH}:/opt/subversion/bin  
PATH=${PATH}:/usr/local/bin  
PATH=${PATH}:/usr/bin  
  
MANPATH=${MANPATH}:/opt/local/share/man
```

```
# Project environment
# -----
#
# # PATH_UKOS_ROOT      = uKOS Project (active workspace)
# # PATH_UKOS_KERNEL    = uKOS kernel

PATH_UKOS_ROOT=${HOME}/DATA/DATA_Private/uKOS_Project/uKOS_Soft
PATH_UKOS_KERNEL=${PATH_UKOS_ROOT}/OS_Kernel-III

# Cross compilation environment
# -----
#
# # PATH_TOOLS_ROOT      = Tool location
# # PATH_TOOLS_GCC        = GCC root location
# # PATH_TOOLS_UNIX       = My BIN UNIX tool location
# # PATH_FTDI_INCLUDES   = FTDI includes
# # PATH_FTDI_LIBRARIES  = FTDI libraries

PATH_TOOLS_ROOT=/opt/uKOS
PATH_TOOLS_GCC=${PATH_TOOLS_ROOT}
PATH_TOOLS_UNIX=${PATH_TOOLS_ROOT}/bin
PATH_FTDI_INCLUDES=${PATH_TOOLS_GCC}/Packages/libftd2xx-1.4.6
PATH_FTDI_LIBRARIES=${PATH_TOOLS_GCC}/Packages/libftd2xx-1.4.6/build

PATH_GCC_ICYFLEX=${PATH_TOOLS_GCC}/cross/gcc-4.6.4/icyflex1
PATH_GCC_COLFIRE=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/coldfire
PATH_GCC_MC6833X=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/cpu32
PATH_GCC_BLACKFIN=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/blackfin
PATH_GCC_CORTEX3=${PATH_TOOLS_GCC}/cross/gcc-7.2.0/arm
PATH_GCC_CORTEX7=${PATH_TOOLS_GCC}/cross/gcc-M7-5.4.1/arm
PATH_GCC_GPUTILS=${PATH_TOOLS_GCC}/cross/gputils-1.5.0-1
PATH_GCC_SDCCPIC=${PATH_TOOLS_GCC}/cross/sdcc-3.6.0
```

```
PATH=${PATH}: ${PATH_TOOLS_UNIX}
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/openocd-0.10.0/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/doxygen-1.8.13/bin
PATH=${PATH}: ${PATH_TOOLS_GCC}/cross/graphviz-2.40.1/bin
PATH=${PATH}: ${PATH_GCC_ICYFLEX}/bin
PATH=${PATH}: ${PATH_GCC_COLFIRE}/bin
PATH=${PATH}: ${PATH_GCC_MC6833X}/bin
PATH=${PATH}: ${PATH_GCC_BLAKFIN}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX3}/bin
PATH=${PATH}: ${PATH_GCC_CORTEX7}/bin
PATH=${PATH}: ${PATH_GCC_GPUTILS}/bin
PATH=${PATH}: ${PATH_GCC_SDCCPIC}/bin

export PATH_FTDI_INCLUDES
export PATH_FTDI_LIBRARIES
export PATH_TOOLS_GCC
export PATH_TOOLS_UNIX
export PATH_UKOS_KERNEL
export PATH_GCC_ICYFLEX
export PATH_GCC_COLFIRE
export PATH_GCC_MC6833X
export PATH_GCC_BLAKFIN
export PATH_GCC_CORTEX3
export PATH_GCC_CORTEX7
export PATH_GCC_GPUTILS
export PATH_GCC_SDCCPIC

export MANPATH
export PATH
```

Put at the same level all the packages: **gcc-4.6.4**, **gcc-7.2.0**, **gdb-8.0.0**, **binutils-2.29**, **newlib-2.5.0** ...

In my case I created **\$(PATH_TOOLS_GCC)/Packages** and I put all the downloaded packages.

F.1.3. Setting-up the compiler (t-arm-elf)

Working with μKOS imposes to have a compiler capable to deal with **multilibs** (**normal** and **fpu** ones). The description of the compiler behavior is done by the file **t-arm-elf**. This takes in account the generation of all the necessary libraries for all the selected targets. This is automatically done by a patch during the creation of the cross-compiler for arm.

The file **t-arm-elf** located at **\${PATH_TOOLS_GCC}/Packages/gcc-7.1.0/gcc/config/arm/** should be modified as followed:

```
MULTILIB_OPTIONS      = marm/mthumb
MULTILIB_DIRNAMES    = arm thumb

# Only we don't actually want any ARM libraries. Or
# vanilla thumb libraries.
MULTILIB_EXCEPTIONS  = marm* mthumb

# Build with any one of arm7tdmi, M3 or M4 support.
MULTILIB_OPTIONS     += mcpu=arm7tdmi-s/mcpu=cortex-m3/mcpu=cortex-m4
MULTILIB_DIRNAMES    += arm7tdmi-s cortex-m3 cortex-m4

# These don't make any sense without thumb, because GCC likes to
# tell you that you have to supply another commandline argument
# rather than just setting it itself.
MULTILIB_EXCEPTIONS += mcpu=arm7tdmi-s* mcpu=cortex-m3* mcpu=cortex-m4*

# All this just to get mfloat-abi=hard mfpu=fpv4-sp-d16 only specified for the M4
MULTILIB_OPTIONS     += mfloat-abi=hard mfpu=fpv4-sp-d16
MULTILIB_DIRNAMES    += float-abi-hard fpuv4-sp-d16
MULTILIB_EXCEPTIONS += mfloat* mthumb/mfloat*
MULTILIB_EXCEPTIONS += mfpu* mthumb/mfpu*
MULTILIB_EXCEPTIONS += mthumb/mcpu=cortex-m4/mfloat-abi=hard
MULTILIB_EXCEPTIONS += mthumb/mcpu=cortex-m4/mfpu=fpv4-sp-d16
MULTILIB_EXCEPTIONS += *arm7tdmi-s*mfloat-abi* *arm7tdmi-s*mfpu*
MULTILIB_EXCEPTIONS += *cortex-m3*mfloat-abi* *cortex-m3*mfpu*

EXTRA_MULTILIB_PARTS = crtbegin.o crtend.o crt.i.o crtn.o
```

F.1.4. Building the packages in this order

1. Launch a terminal.
2. Enter into the Scripts folder: **cd /opt/uKOS/Scripts**.
3. Build the OS X native gcc toolchain: **sh ukos OSX_native.sh**.
4. Build the cpu32 toolchain: **sh ukos_cpu32.sh**.
5. Build the coldfire toolchain: **sh ukos_coldfire2.sh**.
6. Build the blackfin toolchain: **sh ukos_bfin.sh**.
7. Build the ARM cortex toolchain: **sh ukos_arm_eabi.sh**.
8. Build the CSEM icyflex1 toolchain: **sh ukos_icfx1.sh**.
9. Build the D2XX driver: **sh ukos_D2XX.sh**.
10. Build the openocd: **sh ukos_openocd.sh**.
11. Build the boost: **sh ukos_boost_libraries.sh**.
12. Build the gputils: **sh ukos_gputils.sh**.
13. Build the sdcc: **sh ukos_sdcc.sh**.
14. Build the pic: **sh ukos_pic.sh**.
15. Build the graphviz: **sh ukos_graphviz.sh**.
16. Build the doxygen: **sh ukos_doxygen.sh**.

F.1.5. Script for building OSX native gcc compilers

```
# ukos OSX_native.
# =====

#-----
# Author:      Franz Edo.  The 2006-06-28
# Modifs:
#
# SVN:
# $Author::: efr           $: Author of last commit
# $Rev::: 57              $: Revision of last commit
# $Date::: 2016-10-05 22:03:25#$: Date of last commit
#
# Project:     uKOS
# Goal:        Toolchain for generating a native gcc 7.2.0 for OSX
#
#          Usage:
#          sh ukos OSX_native.sh
#
# OS:          OSX 10.12      yes
#               Ubuntu 16.04 LTS   yes
#
# (c) 1992-2017, Franz Edo.
# -----
#
# Franz Edo.
# 5-Route de Cheseaux
# CH 1400 Cheseaux-Noréaz
# edo.franzi@ukos.ch
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
# -----
```

```
if [ -z "$PATH_TOOLS_GCC" ]; then
    echo "Variable PATH_TOOLS_GCC is not set!"
    exit 1
fi

# Choice of the native compiler
# -------

# For OSX > 10.10, the native compiler is LLVM
# For Ubuntu, the native compiler is gcc

case "$(uname)" in
    "Darwin")
        echo "Target OSX"
        export CC=clang
        ;;
    "Linux")
        echo "Target Linux"
        ;;
esac

# Target
# -----

export MACHINE=osx

# Packages
# -----

export GCC_VER=7.2.0

# Environment
# -----

export PACKS_GCC=${PATH_TOOLS_GCC}/Packages/gcc-${GCC_VER}
export BUILD=${PATH_TOOLS_GCC}/builds/gcc-${GCC_VER}
export CROSS=${PATH_TOOLS_GCC}/cross/gcc-${GCC_VER}

export prefix=${CROSS}/${MACHINE}
export executables=${prefix}/bin

# Configurations
# -----

export GCC_CONFIG=" \
--enable-checking=release \
--enable-languages=c,c++,fortran \
--program-suffix=-7.2.0"
```

```
# Building the toolchain
# -----
rm -rf ${BUILD}/${MACHINE}
rm -rf ${CROSS}/${MACHINE}

PATH_SCRIPTS=$(cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )

cd ${PATH_TOOLS_GCC}/Packages
echo "Start of build:" > osx.txt
date >> osx.txt

cd ${PACKS_GCC}
./contrib/download_prerequisites

mkdir -p ${BUILD}/${MACHINE}/gcc-${GCC_VER}
cd ${BUILD}/${MACHINE}/gcc-${GCC_VER}
${PACKS_GCC}/configure \
--prefix=${prefix} \
${GCC_CONFIG} || { echo "Error configuring gcc"; exit 1; }
make -j 8           || { echo "Error building gcc";   exit 1; }
make install       || { echo "Error installing gcc"; exit 1; }

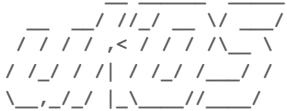
cd ${PATH_TOOLS_GCC}/Packages
echo "End of build:" >> osx.txt
date >> osx.txt
mv osx.txt osx_ready.txt
```

F.1.6. Script for building cpu32 cross-compilers

```
#!/bin/bash

# ukos_cpu32.
# =====

#-----
# Author:      Franzi Edo.   The 2006-06-28
# Modifs:
#
# SVN:
# $Author:: efr          $: Author of last commit
# $Rev:: 48             $: Revision of last commit
# $Date:: 2016-06-25 22:02:38$: Date of last commit
#
# Project:     uKOS
# Goal:        Toolchain for generating generic gcc cross compilers
#               for UNIX like machines (for the uKOS project)
#
#               cpu32 family
#
# Usage:
# sh ukos_cpu32.sh
#
# OS:          OSX 10.12      yes
#               Ubuntu 16.04 LTS   yes
#
# For Xcode version bigger than 6.0.0 it is necessary
# to add the following lines to the script.
#
#       export CC=clang (for OSX 10.10)
#       export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0 (for OSX > 10.10)
#
# (c) 1992-2017, Franzi Edo.
# -----
#
# Franzi Edo.
# 5-Route de Cheseaux
# CH 1400 Cheseaux-Noréaz
# edo.franzi@ukos.ch
```



```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
# -----
if [ -z "$PATH_TOOLS_GCC" ]; then
    echo "Variable PATH_TOOLS_GCC is not set!"
    exit 1
fi

# Choice of the native compiler
# -----
# For OSX == 10.10, the native compiler is LLVM
# For OSX > 10.10, the native compiler is gcc-7.2.0
# For Ubuntu, the native compiler is gcc

case "$(uname)" in
    "Darwin")
        case "$OSTYPE" in
            "darwin14")
                echo "Target OSX clang"
                export CC=clang
                ;;
            "darwin15")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
            "darwin16")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
        esac
        ;;
    "Linux")
        echo "Target Linux"
        ;;
esac
```

```
# Target
# -----

export TARGET=m68k-elf
export MACHINE=cpu32

# Packages
# -----

export BIN_VER=2.29
export GCC_VER=7.2.0
export NLB_VER=2.5.0
export GDB_VER=8.0.0

# Environment
# -----

export PACKS_GCC=${PATH_TOOLS_GCC}/Packages/gcc-${GCC_VER}
export PACKS_BIN=${PATH_TOOLS_GCC}/Packages/binutils-${BIN_VER}
export PACKS_NBL=${PATH_TOOLS_GCC}/Packages/newlib-${NLB_VER}
export PACKS_GDB=${PATH_TOOLS_GCC}/Packages/gdb-${GDB_VER}

export BUILD=${PATH_TOOLS_GCC}/builds/gcc-${GCC_VER}
export CROSS=${PATH_TOOLS_GCC}/cross/gcc-${GCC_VER}

export prefix=${CROSS}/${MACHINE}
export executables=${prefix}/bin

# Configurations
# -----

export BIN_CONFIG=""
export GCC_CONFIG=" \
    --enable-languages=c \
    --disable-libgloss \
    --with-system-zlib"
export NLB_CONFIG=" \
    --enable-newlib-io-long-long \
    --enable-newlib-io-float \
    --disable-libgloss \
    --without-fp"
export GDB_CONFIG=""
```

```
# Building the toolchain
# -----
rm -rf ${BUILD}/${MACHINE}
rm -rf ${CROSS}/${MACHINE}

PATH_SCRIPTS=$(cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

cd ${PATH_TOOLS_GCC}/Packages
echo "Start of build:" > cpu32_temp.txt
date >> cpu32_temp.txt

cd ${PACKS_GCC}
./contrib/download_prerequisites

${PATH_SCRIPTS}/_patch_gcc.sh

${PATH_SCRIPTS}/_build_binutils.sh
${PATH_SCRIPTS}/_build_gcc.sh
${PATH_SCRIPTS}/_build_newlib.sh
${PATH_SCRIPTS}/_build_gdb.sh

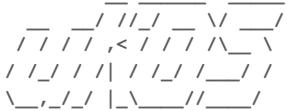
cd ${PATH_TOOLS_GCC}/Packages
echo "End of build:" >> cpu32_temp.txt
date >> cpu32_temp.txt
mv cpu32_temp.txt cpu32_ready.txt
```

F.1.7. Script for building coldfire cross-compilers

```
#!/bin/bash

# ukos_coldfire.
# =====

#-----
# Author:      Franzi Edo.  The 2006-06-28
# Modifs:
#
# SVN:
# $Author::: efr          $: Author of last commit
# $Rev::: 48             $: Revision of last commit
# $Date::: 2016-06-25 22:02:38#$: Date of last commit
#
# Project:    uKOS
# Goal:       Toolchain for generating generic gcc cross compilers
#              for UNIX like machines (for the uKOS project)
#
#           coldfire family
#
# Usage:
# sh ukos_coldfire.sh
#
# OS:        OSX 10.12      yes
#           Ubuntu 16.04 LTS   yes
#
# For Xcode version bigger than 6.0.0 it is necessary
# to add the following lines to the script.
#
#     export CC=clang (for OSX 10.10)
#     export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0 (for OSX > 10.10)
#
# (c) 1992-2017, Franzi Edo.
# -----
#
# Franzi Edo.
# 5-Route de Cheseaux
# CH 1400 Cheseaux-Noréaz
# edo.franzi@ukos.ch
```



```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
# -----
if [ -z "$PATH_TOOLS_GCC" ]; then
    echo "Variable PATH_TOOLS_GCC is not set!"
    exit 1
fi

# Choice of the native compiler
# -----
# For OSX == 10.10, the native compiler is LLVM
# For OSX > 10.12, the native compiler is gcc-7.2.0
# For Ubuntu, the native compiler is gcc

case "$(uname)" in
    "Darwin")
        case "$OSTYPE" in
            "darwin14")
                echo "Target OSX clang"
                export CC=clang
                ;;
            "darwin15")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
            "darwin16")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
        esac
        ;;
    "Linux")
        echo "Target Linux"
        ;;
esac
```

```
# Target
# -----

export TARGET=m68k-elf
export MACHINE=coldfire

# Packages
# -----

export BIN_VER=2.29
export GCC_VER=7.2.0
export NLB_VER=2.5.0
export GDB_VER=8.0.0

# Environment
# -----

export PACKS_GCC=${PATH_TOOLS_GCC}/Packages/gcc-${GCC_VER}
export PACKS_BIN=${PATH_TOOLS_GCC}/Packages/binutils-${BIN_VER}
export PACKS_NBL=${PATH_TOOLS_GCC}/Packages/newlib-${NLB_VER}
export PACKS_GDB=${PATH_TOOLS_GCC}/Packages/gdb-${GDB_VER}

export BUILD=${PATH_TOOLS_GCC}/builds/gcc-${GCC_VER}
export CROSS=${PATH_TOOLS_GCC}/cross/gcc-${GCC_VER}

export prefix=${CROSS}/${MACHINE}
export executables=${prefix}/bin

# Configurations
# -----

export BIN_CONFIG=""
export GCC_CONFIG=" \
    --enable-languages=c \
    --disable-libgloss \
    --with-system-zlib"
export NLB_CONFIG=" \
    --enable-newlib-io-long-long \
    --enable-newlib-io-float \
    --disable-libgloss \
    --without-fp"
export GDB_CONFIG=""
```

```
# Building the toolchain
# -----
rm -rf ${BUILD}/${MACHINE}
rm -rf ${CROSS}/${MACHINE}

PATH_SCRIPTS=$(cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

cd ${TOOLS_GCC}/Packages
echo "Start of build:" > coldfire_temp.txt
date >> coldfire_temp.txt

cd ${PACKS_GCC}
./contrib/download_prerequisites

${PATH_SCRIPTS}/_patch_gcc.sh

${PATH_SCRIPTS}/_build_binutils.sh
${PATH_SCRIPTS}/_build_gcc.sh
${PATH_SCRIPTS}/_build_newlib.sh
${PATH_SCRIPTS}/_build_gdb.sh

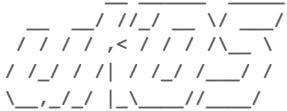
cd ${TOOLS_GCC}/Packages
echo "End of build:" >> coldfire_temp.txt
date >> coldfire_temp.txt
mv coldfire_temp.txt coldfire_ready.txt
```

F.1.8. Script for building blackfin cross-compilers

```
#!/bin/bash

# ukos_bfin.
# =====

#-----
# Author:      Franzi Edo.  The 2006-06-28
# Modifs:
#
# SVN:
# $Author::: efr          $: Author of last commit
# $Rev::: 48             $: Revision of last commit
# $Date::: 2016-06-25 22:02:38#$: Date of last commit
#
# Project:    uKOS
# Goal:       Toolchain for generating generic gcc cross compilers
#              for UNIX like machines (for the uKOS project)
#
#               Blackfin family
#
# Usage:
# sh ukos_bfin.sh
#
# OS:         OSX 10.12      yes
#             Ubuntu 16.04 LTS   yes
#
# For Xcode version bigger than 6.0.0 it is necessary
# to add the following lines to the script.
#
#     export CC=clang (for OSX 10.10)
#     export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0 (for OSX > 10.10)
#
# (c) 1992-2017, Franzi Edo.
# -----
#
# Franzi Edo.
# 5-Route de Cheseaux
# CH 1400 Cheseaux-Noréaz
# edo.franzi@ukos.ch
```



```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
#-----
if [ -z "$PATH_TOOLS_GCC" ]; then
    echo "Variable PATH_TOOLS_GCC is not set!"
    exit 1
fi

# Choice of the native compiler
# -----
# For OSX == 10.10, the native compiler is LLVM
# For OSX > 10.10, the native compiler is gcc-7.2.0
# For Ubuntu, the native compiler is gcc

case "$(uname)" in
    "Darwin")
        case "$OSTYPE" in
            "darwin14")
                echo "Target OSX clang"
                export CC=clang
                ;;
            "darwin15")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
            "darwin16")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
        esac
        ;;
    "Linux")
        echo "Target Linux"
        ;;
esac
```

```
# Target
# -----

export TARGET=bfm-elf
export MACHINE=blackfin

# Packages
# -----

export BIN_VER=2.29
export GCC_VER=7.2.0
export NLB_VER=2.5.0
export GDB_VER=8.0.0

# Environment
# -----

export PACKS_GCC=${PATH_TOOLS_GCC}/Packages/gcc-${GCC_VER}
export PACKS_BIN=${PATH_TOOLS_GCC}/Packages/binutils-${BIN_VER}
export PACKS_NBL=${PATH_TOOLS_GCC}/Packages/newlib-${NLB_VER}
export PACKS_GDB=${PATH_TOOLS_GCC}/Packages/gdb-${GDB_VER}

export BUILD=${PATH_TOOLS_GCC}/builds/gcc-${GCC_VER}
export CROSS=${PATH_TOOLS_GCC}/cross/gcc-${GCC_VER}

export prefix=${CROSS}/${MACHINE}
export executables=${prefix}/bin

# Configurations
# -----

export BIN_CONFIG=""
export GCC_CONFIG=" \
    --enable-languages=c \
    --disable-libgloss \
    --with-system-zlib"
export NLB_CONFIG=" \
    --enable-newlib-io-long-long \
    --enable-newlib-io-float \
    --disable-libgloss"
export GDB_CONFIG=""
```

```
# Building the toolchain
# -----
rm -rf ${BUILD}/${MACHINE}
rm -rf ${CROSS}/${MACHINE}

PATH_SCRIPTS=$(cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

cd ${TOOLS_GCC}/Packages
echo "Start of build:" > bfin_temp.txt
date >> bfin_temp.txt

cd ${PACKS_GCC}
./contrib/download_prerequisites

${PATH_SCRIPTS}/_patch_gcc.sh

${PATH_SCRIPTS}/_build_binutils.sh
${PATH_SCRIPTS}/_build_gcc.sh
${PATH_SCRIPTS}/_build_newlib.sh
${PATH_SCRIPTS}/_build_gdb.sh

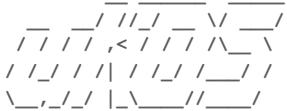
cd ${TOOLS_GCC}/Packages
echo "End of build:" >> bfin_temp.txt
date >> bfin_temp.txt
mv bfin_temp.txt bfin_ready.txt
```

F.1.9. Script for building ARM cross-compilers (M3 - M4 - M7)

```
#!/bin/bash

# ukos_arm_eabi.
# =====

#-----
# Author:      Franzi Edo.  The 2006-06-28
# Modifs:
#
# SVN:
# $Author:: efr          $: Author of last commit
# $Rev:: 48             $: Revision of last commit
# $Date:: 2016-06-25 22:02:38$: Date of last commit
#
# Project:    uKOS
# Goal:       Toolchain for generating generic gcc cross compilers
#              for UNIX like machines (for the uKOS project)
#
#              ARM (cortex M3-M4) family
#
# Usage:
# sh ukos_arm_eabi.sh
#
# OS:         OSX 10.12      yes
#             Ubuntu 16.04 LTS   yes
#
# For Xcode version bigger than 6.0.0 it is necessary
# to add the following lines to the script.
#
#     export CC=clang (for OSX 10.10)
#     export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0 (for OSX > 10.10)
#
# (c) 1992-2017, Franzi Edo.
# -----
#
# Franzi Edo.
# 5-Route de Cheseaux
# CH 1400 Cheseaux-Noréaz
# edo.franzi@ukos.ch
```



```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
# -----
if [ -z "$PATH_TOOLS_GCC" ]; then
    echo "Variable PATH_TOOLS_GCC is not set!"
    exit 1
fi

# Choice of the native compiler
# -----
# For OSX == 10.10, the native compiler is LLVM
# For OSX > 10.10, the native compiler is gcc-7.2.0
# For Ubuntu, the native compiler is gcc

case "$(uname)" in
    "Darwin")
        case "$OSTYPE" in
            "darwin14")
                echo "Target OSX clang"
                export CC=clang
                ;;
            "darwin15")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
            "darwin16")
                echo "Target OSX gcc-7.2.0"
                export CC=${PATH_GCC_CNATIVE}/bin/gcc-7.2.0
                ;;
        esac
        ;;
    "Linux")
        echo "Target Linux"
        ;;
esac
```

```
# Target
# -----

export TARGET=arm-none-eabi
export MACHINE=arm

# Packages
# -----

export BIN_VER=2.29
export GCC_VER=7.2.0
export NLB_VER=2.5.0
export GDB_VER=8.0.0

# Environment
# -----

export PACKS_GCC=${PATH_TOOLS_GCC}/Packages/gcc-${GCC_VER}
export PACKS_BIN=${PATH_TOOLS_GCC}/Packages/binutils-${BIN_VER}
export PACKS_NBL=${PATH_TOOLS_GCC}/Packages/newlib-${NLB_VER}
export PACKS_GDB=${PATH_TOOLS_GCC}/Packages/gdb-${GDB_VER}

export BUILD=${PATH_TOOLS_GCC}/builds/gcc-${GCC_VER}
export CROSS=${PATH_TOOLS_GCC}/cross/gcc-${GCC_VER}

export prefix=${CROSS}/${MACHINE}
export executables=${prefix}/bin

# Configurations
# -----

export BIN_CONFIG=" \
    --enable-interwork"
export GCC1_CONFIG=" \
    --enable-interwork \
    --enable-languages=c,c++ \
    --with-newlib \
    --with-headers=${PACKS_NBL}/newlib/libc/include \
    --with-system-zlib \
    CC=clang"
export NLB_CONFIG=" \
    --enable-interwork \
    --enable-newlib-io-long-long \
    --enable-newlib-io-float"
export GDB_CONFIG=" \
    --enable-interwork"
```

```
# Building the toolchain
# -----
rm -rf ${BUILD}/${MACHINE}
rm -rf ${CROSS}/${MACHINE}

PATH_SCRIPTS=$(cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

cd ${TOOLS_GCC}/Packages
echo "Start of build:" > arm_temp.txt
date >> arm_temp.txt

cd ${PACKS_GCC}
./contrib/download_prerequisites

${PATH_SCRIPTS}/_patch_gcc.sh
${PATH_SCRIPTS}/_patch_cortex.sh

${PATH_SCRIPTS}/_build_binutils.sh
${PATH_SCRIPTS}/_build_gcc_pass1.sh
${PATH_SCRIPTS}/_build_newlib.sh
${PATH_SCRIPTS}/_build_gcc_pass2.sh
${PATH_SCRIPTS}/_build_gdb.sh

cd ${TOOLS_GCC}/Packages
echo "End of build:" >> arm_temp.txt
date >> arm_temp.txt
mv arm_temp.txt arm_ready.txt
```

F.1.10. Script for building icyflex1 cross-compilers

```
#!/bin/bash

# ukos_icfx1.
# =====

#-----
# Author:      Franzi Edo.  The 2006-06-28
# Modifs:
#
# SVN:
# $Author::: efr           $: Author of last commit
# $Rev::: 45              $: Revision of last commit
# $Date::: 2016-03-12 09:53:19#$: Date of last commit
#
# Project:    uKOS
# Goal:       Toolchain for generating generic gcc cross compilers
#             for UNIX like machines (for the uKOS project)
#
#           icyflex-1 family
#
# Usage:
# sh ukos_icfx1.sh
#
# OS:        OSX 10.12      yes
#             Ubuntu 16.04 LTS   yes
#
# For Xcode version bigger than 6.0.0 it is necessary
# to add the following lines to the script.
#
# export CC=clang
#
```

```
#      (c) 1992-2017, Franzi Edo.
# -----
#
#      Franzi Edo.          _ _/_ / / \ \_ / /
#      5-Route de Cheseaux   / / / , < / / / / \_ \
#      CH 1400 Cheseaux-Noréaz / / / / | / / / / \_ / /
#                                \_,/_ / \_\_// \_ / /
#      edo.franzi@ukos.ch
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
# -----
if [ -z "$PATH_TOOLS_GCC" ]; then
    echo "Variable PATH_TOOLS_GCC is not set!"
    exit 1
fi

# Choice of the native compiler
# -----
# For OSX > 10.10, the native compiler is LLVM
# For Ubuntu, the native compiler is gcc

case "$(uname)" in
    "Darwin")
        echo "Target OSX"
        export CC=clang
        ;;
    "Linux")
        echo "Target Linux"
        ;;
esac
```

```
# Target
# -----

export TARGET=icfx1
export MACHINE=icyflex1

# Packages
# -----

export BIN_VER=2.20
export GCC_VER=4.6.4
export NLB_VER=1.18.0

# Environment
# -----

export PACKS_GCC=${PATH_TOOLS_GCC}/Packages/gcc-${GCC_VER}
export PACKS_BIN=${PATH_TOOLS_GCC}/Packages/binutils-${BIN_VER}
export PACKS_NBL=${PATH_TOOLS_GCC}/Packages/newlib-${NLB_VER}

export BUILD=${PATH_TOOLS_GCC}/builds/gcc-${GCC_VER}
export CROSS=${PATH_TOOLS_GCC}/cross/gcc-${GCC_VER}

export prefix=${CROSS}/${MACHINE}
export executables=${prefix}/bin

# Configurations
# -----

export BIN_CONFIG=" \
    --program-prefix=icfx1-"
export GCC_CONFIG=" \
    --program-prefix=icfx1- \
    --enable-languages=c \
    --disable-libssp \
    --disable-libgloss \
    --with-system-zlib"
export NLB_CONFIG=" \
    --program-prefix=icfx1- \
    --enable-newlib-io-long-long \
    --disable-newlib-io-float"
```

```
# Building the toolchain
# -----
rm -rf ${BUILD}/${MACHINE}
rm -rf ${CROSS}/${MACHINE}

PATH_SCRIPTS=$(cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

cd ${PATH_TOOLS_GCC}/Packages
echo "Start of build:" > icfx1_temp.txt
date >> icfx1_temp.txt

cd ${PACKS_GCC}
./contrib/download_prerequisites

${PATH_SCRIPTS}/_patch_icfx1.sh

${PATH_SCRIPTS}/_build_binutils.sh
${PATH_SCRIPTS}/_build_gcc.sh
${PATH_SCRIPTS}/_build_newlib.sh

cd ${PATH_TOOLS_GCC}/Packages
echo "End of build:" >> icfx1_temp.txt
date >> icfx1_temp.txt
mv icfx1_temp.txt icfx1_ready.txt
```

F.1.11. Intermediate scripts for building the binutils, gcc, newlib and gdb

Patch gcc

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start patch gcc:" > patch_gcc.txt
date >> patch_gcc.txt

cd ${PACKS_GCC}
case "${GCC_VER}" in
    "4.8.5")
        patch -p1 -d . < ${PATCH}/gcc/graphite-blocking.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-clast-to-gimple.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-dependences.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-interchange.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-optimize-isl.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-poly.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-scop-detection.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-sese-to-poly.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite.patch
        ;;
    "4.9.3")
        patch -p1 -d . < ${PATCH}/gcc/graphite-blocking.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-clast-to-gimple.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-dependences.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-interchange.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-optimize-isl.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-poly.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-scop-detection.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite-sese-to-poly.patch
        patch -p1 -d . < ${PATCH}/gcc/graphite.patch
        ;;
esac

cd ${PATH_TOOLS_GCC}/Packages
echo "End patch gcc:" >> patch_gcc.txt
date >> patch_gcc.txt
mv patch_gcc.txt patch_gcc_ready.txt
```

Patch cortex

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start patch cortex:" > patch_cortex.txt
date >> patch_gcc_cortex.txt

cd ${PACKS_GCC}
patch -p1 -d . < ${PATCH}/cortex/gcc-cortex-${GCC_VER}.patch

cd ${PATH_TOOLS_GCC}/Packages
echo "End patch gcc:" >> patch_cortex.txt
date >> patch_cortex.txt
mv patch_cortex.txt patch_cortex_ready.txt
```

Patch icfx1

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start patch icfx1:" > patch_icfx1.txt
date >> patch_icfx1.txt

cd ${PACKS_BIN}
sed -i -e 's/@colophon@@colophon/' \
-e 's/doc@cygnus.com/doc@@cygnus.com/' bfd/doc/bfd.texinfo

cd ${PATH_TOOLS_GCC}/Packages
echo "End patch icfx1:" >> patch_icfx1.txt
date >> patch_icfx1.txt
mv patch_icfx1.txt patch_icfx1_ready.txt
```

Binutils

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start building:" > binutils.txt
date >> binutils.txt

mkdir -p ${BUILD}/${MACHINE}/binutils-${BIN_VER}
cd ${BUILD}/${MACHINE}/binutils-${BIN_VER}
${PACKS_BIN}/configure \
    --target=${TARGET} \
    --prefix=${prefix} \
    --enable-multilib \
    --disable-werror \
    --disable-nls \
    --disable-libssp \
    ${BIN_CONFIG} || { echo "Error configuring binutils"; exit 1; }
make -j 4           || { echo "Error building binutils"; exit 1; }
make install       || { echo "Error installing binutils"; exit 1; }
make clean         || { echo "Error cleaning binutils"; exit 1; }

cd ${PATH_TOOLS_GCC}/Packages
echo "End building:" >> binutils.txt
date >> binutils.txt
mv binutils.txt binutils_ready.txt
```

gcc pass 1 (for ARM)

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start building:" > gcc_pass1.txt
date >> gcc_pass1.txt

mkdir -p ${BUILD}/${MACHINE}/gcc-${GCC_VER}
cd ${BUILD}/${MACHINE}/gcc-${GCC_VER}
case "$(uname)" in
    "Darwin")
        CFLAGS="-O2 -fbracket-depth=1024 -pipe"
        CXXFLAGS="-O2 -fbracket-depth=1024 -pipe"
        ${PACKS_GCC}/configure \
            --target=${TARGET} \
            --includedir=/usr/include \
            --prefix=${prefix} \
            --enable-multilib \
            --disable-werror \
            --disable-nls \
            --disable-libssp \
            ${GCC1_CONFIG} || { echo "Error configuring gcc pass 1"; exit 1; }
        make CXXFLAGS="-fbracket-depth=1024" all-gcc -j 4 \
            || { echo "Error building gcc pass 1"; exit 1; }
        make install-gcc || { echo "Error installing gcc pass 1"; exit 1; }
    ;;
    "Linux")
        ${PACKS_GCC}/configure \
            --target=${TARGET} \
            --includedir=/usr/include \
            --prefix=${prefix} \
            --enable-multilib \
            --disable-werror \
            --disable-nls \
            --disable-libssp \
            ${GCC1_CONFIG} || { echo "Error configuring gcc pass 1"; exit 1; }
        make all-gcc -j 4 || { echo "Error building gcc pass 1"; exit 1; }
        make install-gcc || { echo "Error installing gcc pass 1"; exit 1; }
    ;;
esac

cd ${PATH_TOOLS_GCC}/Packages
echo "End building:" >> gcc_pass1.txt
date >> gcc_pass1.txt
mv gcc_pass1.txt gcc_pass1_ready.txt
```

gcc pass 2 (for ARM)

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start building:" > gcc_pass2.txt
date >> gcc_pass2.txt

cd ${BUILD}/${MACHINE}/gcc-${GCC_VER}
case "$(uname)" in
    "Darwin")
        make CXXFLAGS="-fbracket-depth=1024" -j 4 \
            || { echo "Error building gcc pass 2"; exit 1; }
        make install || { echo "Error installing gcc pass 2"; exit 1; }
        make clean   || { echo "Error cleaning gcc pass 2"; exit 1; }
    ;;
    "Linux")
        make -j 4     || { echo "Error building gcc pass 2"; exit 1; }
        make install || { echo "Error installing gcc pass 2"; exit 1; }
        make clean   || { echo "Error cleaning gcc pass 2"; exit 1; }
    ;;
esac

cd ${PATH_TOOLS_GCC}/Packages
echo "End building:" >> gcc_pass2.txt
date >> gcc_pass2.txt
mv gcc_pass2.txt gcc_pass2_ready.txt
```

gcc (for all the CPUs)

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start building:" > gcc.txt
date >> gcc.txt

mkdir -p ${BUILD}/${MACHINE}/gcc-${GCC_VER}
cd ${BUILD}/${MACHINE}/gcc-${GCC_VER}
${PACKS_GCC}/configure \
--target=${TARGET} \
--prefix=${prefix} \
--enable-shared \
--enable-multilib \
--enable-target-optspace \
--disable-nls \
--disable-libssp \
--with-gnu-as \
--with-gnu-ld \
${GCC_CONFIG} || { echo "Error configuring gcc"; exit 1; }
make -j 4           || { echo "Error building gcc";   exit 1; }
make install       || { echo "Error installing gcc"; exit 1; }
make clean         || { echo "Error cleaning gcc";  exit 1; }

cd ${PATH_TOOLS_GCC}/Packages
echo "End building:" >> gcc.txt
date >> gcc.txt
mv gcc.txt gcc_ready.txt
```

newlib

```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start building:" > newlib.txt
date >> newlib.txt

mkdir -p ${BUILD}/${MACHINE}/newlib-${NLB_VER}
cd ${BUILD}/${MACHINE}/newlib-${NLB_VER}
${PACKS_NBL}/configure \
--target=${TARGET} \
--prefix=${prefix} \
--enable-multilib \
--disable-werror \
--disable-nls \
--disable-libssp \
${NLB_CONFIG} || { echo "Error configuring newlib"; exit 1; }
make -j 4           || { echo "Error building newlib";   exit 1; }
make install       || { echo "Error installing newlib"; exit 1; }
make clean         || { echo "Error cleaning newlib";  exit 1; }

cd ${PATH_TOOLS_GCC}/Packages
echo "End building:" >> newlib.txt
date >> newlib.txt
mv newlib.txt newlib_ready.txt
```

gdb

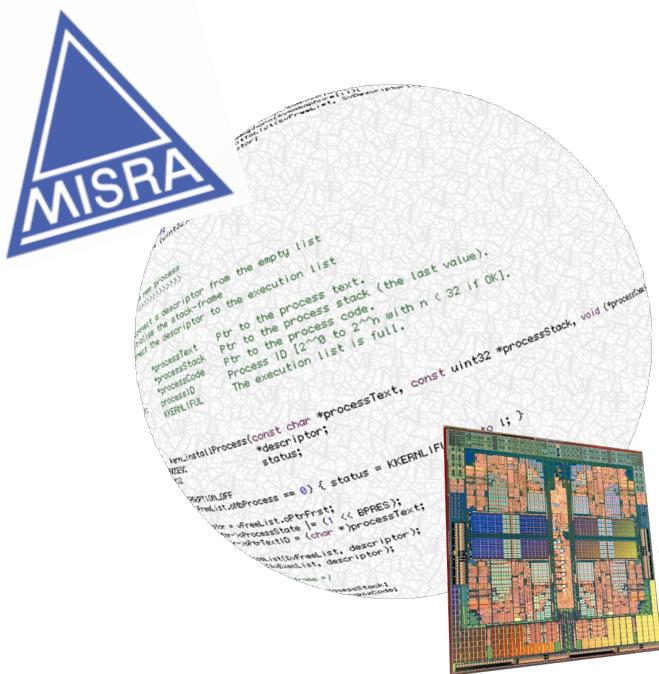
```
cd ${PATH_TOOLS_GCC}/Packages
echo "Start building:" > gdb.txt
date >> gdb.txt

mkdir -p ${BUILD}/${MACHINE}/gdb-${GDB_VER}
cd ${BUILD}/${MACHINE}/gdb-${GDB_VER}
${PACKS_GDB}/configure \
--target=${TARGET} \
--prefix=${prefix} \
--enable-multilib \
--disable-werror \
--disable-nls \
--disable-libssp \
${GDB_CONFIG} || { echo "Error configuring gdb"; exit 1; }
make -j 4          || { echo "Error building gdb";   exit 1; }
make install      || { echo "Error installing gdb"; exit 1; }
make clean         || { echo "Error cleaning gdb";  exit 1; }

cd ${PATH_TOOLS_GCC}/Packages
echo "End building:" >> gdb.txt
date >> gdb.txt
mv gdb.txt gdb_ready.txt
```


G. Annex

Misra - C μKOS-III compliance matrix



G.1. Introduction

The sources of the µKOS project is close to be 100% compliant with the Motor Industry Software Reliability Association (MISRA) C Coding Standards. These standards were created by MISRA to improve the reliability and predictability of C programs in critical automotive systems. Members of the MISRA consortium include Delco Electronics, Ford Motor Company, Jaguar Cars Ltd., Lotus Engineering, Lucas Electronics, Rolls-Royce, Rover Group Ltd., and other firms and universities dedicated to improving safety and reliability in automotive electronics.

All the details of this standard can be obtained directly from the MISRA association web site, <http://www.misra.org.uk>. A detailed µKOS-III compliance matrix describing all of MISRA's C Coding rules is presented (based on the original 1998 rules). Few changes are needed to be made µKOS-III to fully comply with these rules since µKOS-III was designed from the beginning with most of these practices.

G.2. μKOS-III MISRA C 1998 compliance matrix

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
Environment			
1	Required	Yes	All code shall conform to ISO/IEC 9899:1990 “Programming languages — C”, amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2:1996
2	Advisory	Yes	Code written in languages other than C should only be used if there is a defined interface standard for object code to which the compilers/assemblers for both languages conform
3	Advisory	No (1)	Assembly language functions that are called from C should be written as C functions containing only in-line assembly language, and in-line assembly language should not be embedded in normal C code
4	Advisory	Yes	Provision should be made for appropriate run-time checking
Character Sets			
5	Required	Yes	Only those characters and escape sequences which are defined in the ISO C standard shall be used
6	Required	Yes	Values of character types shall be restricted to a defined and documented subset of ISO 10646-1
7	Required	Yes	Trigraphs shall not be used
8	Required	Yes	Multibyte characters and wide string literals shall not be used
Comments			
9	Required	Yes	Comments shall not be nested

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
10	Advisory	Yes	Sections of code should not be 'commented out'
Identifiers			
11	Required	Yes	Identifiers (internal and external) shall not rely on significance of more than 31 characters. Furthermore the compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers
12	Advisory	Yes	No identifier in one name space shall have the same spelling as an identifier in another name space
13	Advisory	Yes	The basic types of char, int, short, long, float and double should not be used, but specific-length equivalents should be typedef'd for the specific compiler, and these type names used in the code
14	Required	Yes	The type char shall always be declared as unsigned char or signed char
15	Advisory	Yes (2)	Floating point implementations should comply with a defined floating point standard
16	Required	Yes (2)	The underlying bit representations of floating point numbers shall not be used in any way by the programmer
17	Required	Yes	typedef names shall not be reused
Constants			
18	Advisory	Yes (3)	Numeric constants should be suffixed to indicate type, where an appropriate suffix is available
19	Required	Yes	Octal constants (other than zero) shall not be used
Declarations and Definitions			

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
20	Required	Yes	All object and function identifiers shall be declared before use
21	Required	Yes	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier
22	Advisory	Yes	Declarations of objects should be at function scope unless a wider scope is necessary
23	Advisory	Yes	All declarations at file scope should be static where possible
24	Required	Yes	Identifiers shall not simultaneously have both internal and external linkage in the same translation unit
25	Required	Yes	An identifier with external linkage shall have exactly one external definition
26	Required	Yes	If objects or functions are declared more than once they shall have compatible declarations
27	Advisory	Yes	External objects should not be declared in more than one file
28	Advisory	Yes	The register storage class specifier should not be used
29	Required	Yes	The use of a tag shall agree with its declaration
Initialisation			
30	Required	Yes	All automatic variables shall have been assigned a value before being used
31	Required	Yes	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
32	Required	Yes	In an enumerator list, the '=' construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised
Operators			
33	Required	Yes	The right hand operand of a && or operator shall not contain side effects
34	Required	Yes	The operands of a logical && or shall be primary expressions
35	Required	Yes	Assignment operators shall not be used in expressions which return Boolean values
36	Advisory	Yes	Logical operators should not be confused with bitwise operators
37	Required	Yes	Bitwise operations shall not be performed on signed integer types
38	Required	Yes	The right hand operand of a shift operator shall lie between 0 and one less than the width in bits of the left hand operand (inclusive)
39	Required	Yes	The unary minus operator shall not be applied to an unsigned expression
40	Required	Yes	The sizeof operator should not be used on expressions that contain side effects
41	Advisory	Yes	The implementation of integer division in the chosen compiler should be determined, documented and taken into account
42	Required	Yes	The comma operator shall not be used, except in the control expression of a for loop

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
Conversions			
43	Required	Yes	Implicit conversions which may result in a loss of information shall not be used
44	Advisory	No	Redundant explicit casts should not be used
45	Required	No (4)	Type casting from any type to or from pointers shall not be used
Expressions			
46	Required	Yes	The value of an expression shall be the same under any order of evaluation that the standard permits
47	Advisory	Yes	No dependence should be placed on C's operator precedence rules in expressions
48	Advisory	Yes	Mixed precision arithmetic should use explicit casting to generate the desired result
49	Advisory	Yes	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean
50	Required	Yes	Floating point variables shall not be tested for exact equality or inequality
51	Required	Yes	Evaluation of constant unsigned integer expressions should not lead to wrap-around
Control Flow			
52	Required	Yes	There shall be no unreachable code
53	Required	Yes	All non-null statements shall have a side-effect
54	Required	Yes	A null statement shall only occur on a line by itself, and shall not have any other text on the same line

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
55	Advisory	Yes	Labels should not be used, except in switch statements
56	Required	Yes	The goto statement shall not be used
57	Required	No	The continue statement shall not be used
58	Required	Yes	The break statement shall not be used (except to terminate the cases of a switch statement)
59	Required	Yes	The statements forming the body of an if, else if, else, while, do while or for statement shall always be enclosed in braces
60	Advisory	No	All if, else if constructs should contain a final else clause
61	Required	Yes	Every non-empty case clause in a switch statement shall be terminated with a break statement
62	Required	Yes	All switch statements should contain a final default clause
63	Advisory	Yes	A switch expression should not represent a Boolean value
64	Required	Yes	Every switch statement shall have at least one case
65	Required	Yes	Floating point variables shall not be used as loop counters
66	Advisory	Yes	Only expressions concerned with loop control should appear within a for statement
67	Advisory	Yes	Numeric variables being used within a for loop for iteration counting should not be modified in the body of the loop

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
Functions			
68	Required	Yes	Functions shall always be declared at file scope
69	Required	Yes	Functions with variable numbers of arguments shall not be used
70	Required	Yes	Functions shall not call themselves, either directly or indirectly
71	Required	Yes	Functions shall always have prototype declarations and the prototype shall be visible at both the function definition and call
72	Required	Yes	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical
73	Required	Yes	Identifiers shall either be given for all of the parameters in a function prototype declaration, or for none
74	Required	Yes	If identifiers are given for any of the parameters, then the identifiers used in the declaration and definition shall be identical
75	Required	Yes	Every function shall have an explicit return type
76	Required	Yes	Functions with no parameters shall be declared with parameter type void
77	Required	Yes	The unqualified type of parameters passed to a function shall be compatible with the unqualified expected types defined in the function prototype
78	Required	Yes	The number of parameters passed to a function shall match the function prototype

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
79	Required	Yes	The values returned by void functions shall not be used
80	Required	Yes	Void expressions shall not be passed as function parameters
81	Advisory	Yes	const qualification should be used on function parameters which are passed by reference, where it is intended that the function will not modify the parameter
82	Advisory	No	A function should have a single point of exit
83	Required	Yes	For functions with non-void return type: i) there shall be one return statement for every exit branch (including the end of the program) , ii) each return shall have an expression, iii) the return expression shall match the declared return type
84	Required	Yes	For functions with void return type, return statements shall not have an expression
85	Advisory	Yes	Functions called with no parameters should have empty parentheses
86	Advisory	No	If a function returns error information, then that error information should be tested
Pre-processing Directives			
87	Required	Yes	#include statements in a file shall only be preceded by other pre- processor directives or comments
88	Required	Yes	Non-standard characters shall not occur in header file names in #include directives

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
89	Required	Yes	"The #include directive shall be followed by either a <filename> or ""filename"" sequence."
90	Required	Yes	C macros shall only be used for symbolic constants, function-like macros, type qualifiers and storage class specifiers
91	Required	Yes	Macros shall not be #define'd and #undef'd within a block
92	Advisory	No	#undef should not be used
93	Advisory	Yes	A function should be used in preference to a function-like macro
94	Required	Yes	A function-like macro shall not be 'called' without all of its arguments
95	Required	Yes	Arguments to a function-like macro shall not contain tokens that look like pre-processing directives
96	Required	Yes	In the definition of a function-like macro the whole definition, and each instance of a parameter, shall be enclosed in parentheses
97	Advisory	Yes	Identifiers in pre-processor directives should be defined before use
98	Required	Yes	There shall be at most one occurrence of the # or ## pre-processor operators in a single macro definition
99	Required	Yes	All uses of the #pragma directive shall be documented and explained
100	Required	Yes	The defined pre-processor operator shall only be used in one of the two standard forms
Pointers and Arrays			

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
101	Advisory	No	Pointer arithmetic should not be used
102	Advisory	Yes	No more than 2 levels of pointer indirection should be used
103	Required	Yes	Relational operators shall not be applied to pointer types except where both operands are of the same type and point to the same array, structure or union
104	Required	Yes	Non-constant pointers to functions shall not be used
105	Required	Yes	All the functions pointed to by a single pointer to function shall be identical in the number and type of parameters and the return type
106	Required	Yes	The address of an object with automatic storage shall not be assigned to an object which may persist after the object has ceased to exist
107	Required	Yes	The null pointer shall not be de-referenced
Structures and Unions			
108	Required	Yes	In the specification of a structure or union type, all members of the structure or union shall be fully specified
109	Required	Yes	Overlapping variable storage shall not be used
110	Required	Yes	Unions shall not be used to access the sub-parts of larger data types
111	Required	Yes	Bit fields shall only be defined to be of type unsigned int or signed int
112	Required	Yes	Bit fields of type signed int shall be at least 2 bits long

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
113	Required	Yes	All the members of a structure (or union) shall be named and shall only be accessed via their name
Standard Libraries			
114	Required	Yes	Reserved words and standard library function names shall not be redefined or undefined
115	Required	Yes	Standard library function names shall not be reused
116	Required	Yes (5)	All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation
117	Required	Yes	The validity of values passed to library functions shall be checked
118	Required	No (6)	Dynamic heap memory allocation shall not be used
119	Required	Yes	The error indicator errno shall not be used
120	Required	Yes	The macro offsetof, in library <stddef.h>, shall not be used
121	Required	Yes	<locale.h> and the setlocale function shall not be used
122	Required	Yes	The setjmp macro and the longjmp function shall not be used
123	Required	Yes	The signal handling facilities of <signal.h> shall not be used
124	Required	Yes	The input/output library <stdio.h> shall not be used in production code
125	Required	Yes	The library functions atof, atoi and atol from library <stdlib.h> shall not be used

μKOS-III RTOS MISRA C 1998 compliance matrix			
Rule number	Required rule	μKOS-III compliant	Brief description of the rule
126	Required	Yes (7)	The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used
127	Required	Yes	The time handling functions of library <time.h> shall not be used

MISRA C 1998		μKOS-III
		Violations
Required rules	97	2 (45, 118)
Advisory rules	30	6 (3, 60, 82, 86, 92, 101)

Notes

- (1) μKOS-III assembly code is encapsulated by C macros located in the macros.h file. The only exception is the inline assembly code used by the μKernel for the context switching, for the exception crash management and for the initialization of the stack pointer in the crt0. For readability reason, it is better to maintain the assembler code “visible” in these part of the project
- (2) μKOS-III uses the gcc implementation
- (3) μKOS-III assumes that the compiler will catch those
- (4) Too restrictive in embedded
- (5) Computer specific
- (6) μKOS-III μKernel does not use any dynamic memory allocation. However, some operating system modules (not mandatory) could use syos_malloc and derivative. In SoC, for footprint memory reasons, it is necessary to dynamically share memory regions; in this case, syos_malloc is unavoidable
- (7) μKOS-III uses its own exit and not the stdlib one

G.3. References

MISRA “Guidelines For The Use Of The C Language In Vehicle Based Software”, April 1998.