



BIOS REFERENCE MANUAL

Documentation author

K-Team S.A.
Ch. de Vuasset, CP 111
1028 Préverenges
Switzerland

email: info@k-team.com
WWW: <http://www.k-team.com>

Trademark Acknowledgements

IBM PC: International Business Machines Corp.
Macintosh: Apple Corp.
SUN Sparc-Station: SUN Microsystems Corp.
LabVIEW: National Instruments Corp.
Khepera: K-Team S.A.

NOTICE:

- The contents of this manual are subject to change without notice.
- All efforts have been made to ensure the accuracy of the contents of this manual. However, should any errors be detected. In this case, please inform the K-Team S.A.
- The above notwithstanding K-Team S.A. can assume no responsibility for any errors in this manual.



Table of content

Table of content	1
Abstract	3
Preliminary	3
BIOS organisation	4
Basic managers	4
General constraints	5
Rules for building applications	5
BIOS	8
Table of content	8
Generalities	9
COM	15
Table of content	15
Generalities	16
TIM	24
Table of content	24
Generalities	25
MOT	44
Table of content	44
Generalities	45
The PID implementation	46
The speed profile generator	47
SENS	64
Table of content	64
Generalities	65
MSG	71
Table of content	71
Generalities	72
MMA	79
Table of content	79
Generalities	80
SER	100
Table of content	100
Generalities	101
VAR	110
Table of content	110
Generalities	111



UIO	124
Table of content	124
Generalities	
CTR	135
Table of content	135
Generalities	136
STR	143
Table of content	143
Generalities	144
References	159

**K-Team**

<http://www.k-team.com>

info@k-team.com

Koala BIOS 1.00 Reference Manual

Rev. 1.00

Abstract

The high level of complexity of the Koala robot coupled with its multi-microcontroller architecture [Fra91] and its multitasking capabilities requires a robust low level software named BIOS (Basic I/Os system). This document describes how this software is organised to manage all the system resources and give all the necessary information to use them for building applications.

Preliminary

The reader is supposed to have a good knowledge of the MC68xxx programming and of MC68331 microcontroller hardware features [Mot89][Mot91][Mot92]. The program examples are shown in C and in assembler CALM¹ syntax [JDN86].

At the end of this document some examples will be shown to make the work with Koala easy.

1. CALM is the abbreviation for Common Assembler Language for Microprocessors which is one product designed at LAMI-EPFL.

BIOS organisation

Figure 1 shows the basic subdivision of the BIOS. Different managers were designed to control only a specific part of the system (e.g. module MOT controls all the motion resources, module MSG controls the network communications, TIM controls the multitasking, etc.).

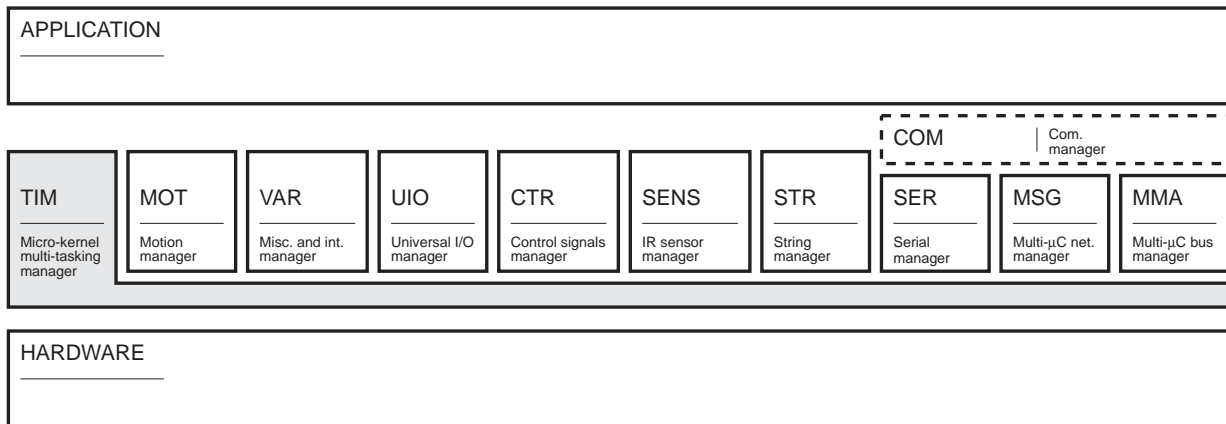


Figure 1: General topology of the BIOS

The code is completely relocatable and is designed to allow an easy interface with a high level language such as C.

Basic managers

As already mentioned, each physical part of the system is under the control of a specific manager. The complete details of these modules will be presented later. Here is the list:

- BIOS: global core of the BIOS.
- MOT: Motor manager.
- SENS: infra-red sensor manager.
- COM: Host communication manager. Uses resources of SER, MSG or MMA.
- SER: serial link manager.
- MSG: multi-microcontroller network communication manager.
- MMA: multi-microcontroller parallel bus communication manager.
- VAR: misc device manager (jumpers, LEDs, etc.).
- UIO: general purpose I/O manager
- CTR: control electronics manager (power supply, power consumption, etc.)
- TIM: multitasking manager.
- STR: string manager.



General constraints

It is vital to observe the few rules below in order to realise robust applications. None of the Koala hardware resources initialised and used by the BIOS should be modified.

- VBR register has to be initialised before using the BIOS. This is done during the start-up process.
- The parameters are stacked before the function calls; their size always takes on 32-bits even if only 8 or 16-bits are significant.
- All the function calls can modify the following microcontroller registers: D0, D1, A0, A1, F. The BIOS never modifies the other registers.
- If one call has to return one result, the register D0 is used.
- BIOS uses TRAP #0 to TRAP #7.

Rules for building applications

All the programs that compose the Koala system as well as the user applications are under the control of micro-kernel (TIM manager). This software architecture allows to run simultaneously as many as 32-tasks. The user who would like to write applications needs to be at ease with multi-tasking programming methodology.

1. Using the BIOS managers

Before using the functions available inside the managers, the user has to initialise them. The BIOS and TIM managers do not need to be initialised.

Here is an example to initialize the SER manager:

```
ser_reset();
...
```

2. Launching and killing processes

The BIOS does not include any memory management. For this reason and only for C applications, the launching and the killing system calls are not directly managed by the TIM manager. An additional C layer (including a couple of new system calls) is present between the application and the BIOS.

Here is an example

```
int32    status;
uint32   id;
static  char    textId[] = "Process to ...\n\r"
...
status = install_task(textId, 800, procedure);
if (status < 0) exit(0); /* Error, ... */
id = (uint32)status;
...
...
status = kill_task(id);
if (status < 0) exit(0); /* Error, ... */
...
```




3. Protection of the critical memory resources

It is sometimes necessary to be sure that a memory structure is protected against some external writing (ex. coming from an other task). To do that, critical resource accesses need to be encapsulated by lock and unlock semaphore. During the encapsulation time, only the active task is executed.

Here is an example

```
/* Writing process */
...
tim_lock();    /* This locks the time sharing */
for (i = 0; i < KNBELEMENTS; i++)
    vector[i] = i;

tim_unlock(); /* This unlocks the time sharing */
...
...
...
/* Reading process */
...
tim_lock();    /* This locks the time sharing */
for (i = 0; i < KNBELEMENTS; i++)
    workVector[i] = vector[i];

tim_unlock(); /* This unlocks the time sharing */
```



4. Protection of the critical I/Os resources

As for the memory protection, I/Os need to be protected. To avoid to lock the time sharing for this purpose, a couple of system calls are used to reserve and to release an I/Os channel. For the user an easier implementation by a macro is available.

Here is an example

```
/* Process 1 */
...
RESERVE_COM    /* This reserves the standard I/Os channel */
printf("K-System © E. Franzi, K-Team S.A.\n");
RELEASE_COM    /* This releases the standard I/Os channel */
...
...
...
/* Process 2 */
...
RESERVE_COM    /* This reserves the standard I/Os channel */
printf("Khepera minirobot\n");
RELEASE_COM    /* This releases the standard I/Os channel */
```



BIOS

Rev. 1.00

K-Team

<http://www.k-team.com>

info@k-team.com

BIOS

BIOS manager

Family ID: 'BIOS'

Table of content

<code>bios_reset()</code>	10
<code>bios_get_ident()</code>	11
<code>bios_get_rev()</code>	12
<code>bios_get_system()</code>	13
<code>bios_restart_system()</code>	14



Generalities

This module includes all the files necessary to manage a dedicated part of the system (“XYZ.ASI”). Moreover, all the input system calls to the different modules are managed by one table located at the beginning of the BIOS code.

The system calls are performed by pushing into the stack the different parameters followed by a “TRAP #0” and a number of 16-bits which codes the call. Obviously, the stack has to be readjusted according to the number of parameters pushed. Here is an example:

```

...
push.32 Value1      ; first parameter
push.32 Value2      ; second parameter
trap    #0          ; BIOS call
.16     CallNumber  ; the number of the function
add.32  #4*2,SP     ; stack adjust
...

```

To improve the readability of the programs, the BIOS system call sequence can be replaced by a macro.

```

.MACRO   CALL_BIOS
trap    #0          ; call the BIOS
.16     %1          ; the number of the function
add.32  #4*%2,SP   ; stack adjust
.ENDMACRO

```

The previous example becomes:

```

...
push.32 Value1      ; first parameter
push.32 Value2      ; second parameter
CALL_BIOS CallNumber,2 ; BIOS call
...

```



bios_reset()

Init the resources of the module. DO NOT USE FOR APPLICATIONS!

This system call inits all the common resources used by the different “BIOS” modules. bios_reset() has to be called before using any other system call. This system call is performed during the start-up.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS bios_reset,0      ; execute the function  
  
...  
bios_reset();
```



bios_get_ident()

Get the pointer on an identifier string.

This system call returns the pointer on an "identifier" ASCII string (terminated with '\0') which indicates the date and the revision number of the "BIOS".

Input (stacking order):

-

Output:

D0 identifier Pointer on the identifier string.

Call examples in assembler and C:

```
...
CALL_BIOS bios_get_ident,0        ; execute the function
move.32    D0,{A6}+identifier    ; pointer on the identifier string

char       *identifier;
...
identifier = bios_get_ident();
```



bios_get_rev()

Get the BIOS version and revision.

This system call returns an ASCII identifier containing the version and revision of the "BIOS".

Input (stacking order):

-

Output:

D0[31..24]	"5"	Version.
D0[23..16]	."	Point.
D0[15..8]	"0"	MSB revision.
D0[7..0]	"0"	LSB revision.

Call examples in assembler and C:

```
...  
CALL_BIOS bios_get_rev,0      ; execute the function  
move.32   D0,{A6}+version    ; version  
  
uint32    version;  
...  
version = bios_get_rev();
```



bios_get_system()

Get the family identifier (type) of the system.

This system call returns the family identifier (type) of the device supported by the K-Team. Here is the list of the available devices:

Input (stacking order):

-

Output:

D0 family Family identifier.

Call examples in assembler and C:

```
...  
CALL_BIOS bios_get_system,0     ; execute the function  
move.32    D0,{A6}+family       ; family identifier  
  
uint32     family;  
...  
family = bios_get_system();
```




`bios_restart_system()`

Perform a restart of the system.

This system call allows to restart the system. All the peripherals, memory and “BIOS” managers are initialised. The function selected by the jumper is executed after this system call.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS bios_restart_system,0; execute the function  
-  
  
...  
bios_restart_system();
```



COM

Rev. 1.02

K-Team

<http://www.k-team.com>

info@k-team.com

COM

COM manager (i/o manager)

Family ID: 'BIOS'

Table of content

com_reset()17
com_reserve_channel()18
com_release_channel()19
com_send_buffer(buffer, size)20
com_receive_byte()21
com_get_status_channel()22



Generalities

This module manages all the I/Os channels of the system. The user must only define (during the start-up) which physical channel is active. The default channel is under the control of the SER manager Here is an example:

```

...
RESERVE_COM                /* if the channel is busy the task is switched */
printf("KOS EFr. 99\n");   /* send by the active channel */
RELEASE_COM                /* release the channel (for other tasks) */
...

```

In this example the string is sent on the active channel. The COM manager redirects the communications according to some conditions during the start-up. Table 1 shows some conditions for the redirection at boot depending from the extension configuration. Other redirections (MMA) can be also used.

SER	Radio turret	Irda turret	Other I/Os turret	Redirection to ...
Active	Not active	Not active	Not active	Channel SER
Active	Active	x	x	Channel Radio
Active	x	Active	x	Channel Irda
Active	Active	Active	x	Channel Radio
Active	Active	Active	Active	Channel Radio

Table 1: Redirection of the I/Os channels



The system calls are performed by pushing into the stack the different parameters followed by a software TRAP #6 and a 16-bit number which codes the call. Obviously, the stack has to be readjusted according to the number of parameters pushed. Here is an example:

```
...
push.32 Value1      ; first parameter
push.32 Value2      ; second parameter
trap      #6          ; COM call
.16       CallNumber ; the number of the function
add.32    #4*2,SP     ; stack adjust
...
```

To improve the readability of the programs, the COM system call sequence can be replaced by a macro.

```
.MACRO    CALL_COM
trap      #6          ; call the COM
.16       %1          ; the number of the function
add.32    #4*%2,SP   ; stack adjust
.ENDMACRO
```

The previous example becomes:

```
...
push.32 Value1      ; first parameter
push.32 Value2      ; second parameter
CALL_COM  CallNumber,2 ; COM call
...
```



`com_reset()`

Init of the resources of the manager.

This system call inits the manager.

Input:

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_COM COM_reset,0 ; execute the function
```

```
...  
com_reset();
```



com_reserve_channel()

Reserve the active channel.

This system call reserves the active channel for a transaction. The active channel is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

-

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	Channel busy.

Call examples in assembler and C:

```
...
CALL_COM  com_reserve_channel,0; execute the function
test.32   D0                      ;
jump,mi   R8^Error                 ; wait ...

int32     status;
...
status = com_reserve_channel();
if (status < 0) return -1;
```



`com_release_channel()`

Release the active channel.

This system call releases the active channel. The other tasks can now use this channel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_COM  com_release_channel,0    ; execute the function  
  
...  
com_release_channel();
```



com_send_buffer(buffer, size)

Send one buffer via the active channel.

This system call sends one buffer of less than 500 bytes by the active channel. An error is returned (if any).

Input (stacking order):

size	Size of the buffer to send.
buffer	Pointer on the buffer.

Output:

D0	0	OK.
D0	-1	Channel busy.
D0	-2	Size of the buffer too big.
D0	-3	Size of the buffer = 0.
D0	-4	Hardware error.

Call examples in assembler and C:

```

...
push.32    {A6}+size           ; size of the buffer to send
push.32    #{A6}+buffer       ; pointer on the buffer
CALL_COM   com_sens_buffer,2   ; execute the function
test.32    D0                  ;
jump,mi    R8^Error           ; channel error

int32      status;
uint8      *buffer;
uint32     size;
...
status = com_sens_buffer(buffer, size);
if (status < 0) return status;

```




com_receive_byte()

Receive one byte via the active channel.

This system call looks for the reception buffer of the active channel if one byte is available.

Input (stacking order):

-

Output:

D0	+16'000000nn	nn = byte.
D0	-1	Buffer empty.
D0	-4	Hardware error.

Call examples in assembler and C:

```
...
CALL_COM  com_receive_byte,0    ; execute the function
test.32   D0                    ;
jump,mi   R8^Error              ; channel error
move.8    D0, {A6}+aByte        ; a character

int32     status;
uint8     aByte;
...
status = com_receive_byte();
if (status < 0) return status;
aByte = (uint8)status;
```



com_get_status_channel()

Get the status of the active channel.

This system call looks for the status of the active channel.

Input (stacking order):

-

Output:

D0	+xxx0000001	Tx buffer not empty
D0	+xxx0000010	Rx buffer not empty
D0	-4	Hardware error.

Call examples in assembler and C:

```
...
CALL_COM  com_get_status_channel,0 ; execute the function
test.32  D0 ;
jump,mi  R8^Error ; hardware error

int32    status;
...
status = com_get_status_channel();
```



TIM

Rev. 1.00

K-Team

<http://www.k-team.com>
info@k-team.com

TIM

TIM manager (multi-tasking kernel)

Family ID: 'BIOS'

Table of content

<code>tim_reset()</code>	.26
<code>tim_new_inst_task(textId, stack, procedure)</code>	.27
<code>tim_remove_inst_task(id)</code>	.28
<code>tim_suspend_task(time)</code>	.29
<code>tim_generate_event()</code>	.30
<code>tim_wait_event(taskMask)</code>	.31
<code>tim_get_id()</code>	.32
<code>tim_get_ticcount()</code>	.33
<code>tim_run_kernel()</code>	.34
<code>tim_switch_fast()</code>	.35
<code>tim_lock()</code>	.36
<code>tim_unlock()</code>	.37
<code>tim_define_association(reference, general)</code>	.38
<code>tim_find_association(reference)</code>	.39
<code>tim_remove_association(reference)</code>	.40
<code>tim_wait_sync(syncMask)</code>	.41
<code>tim_get_task_des_ptr()</code>	.42

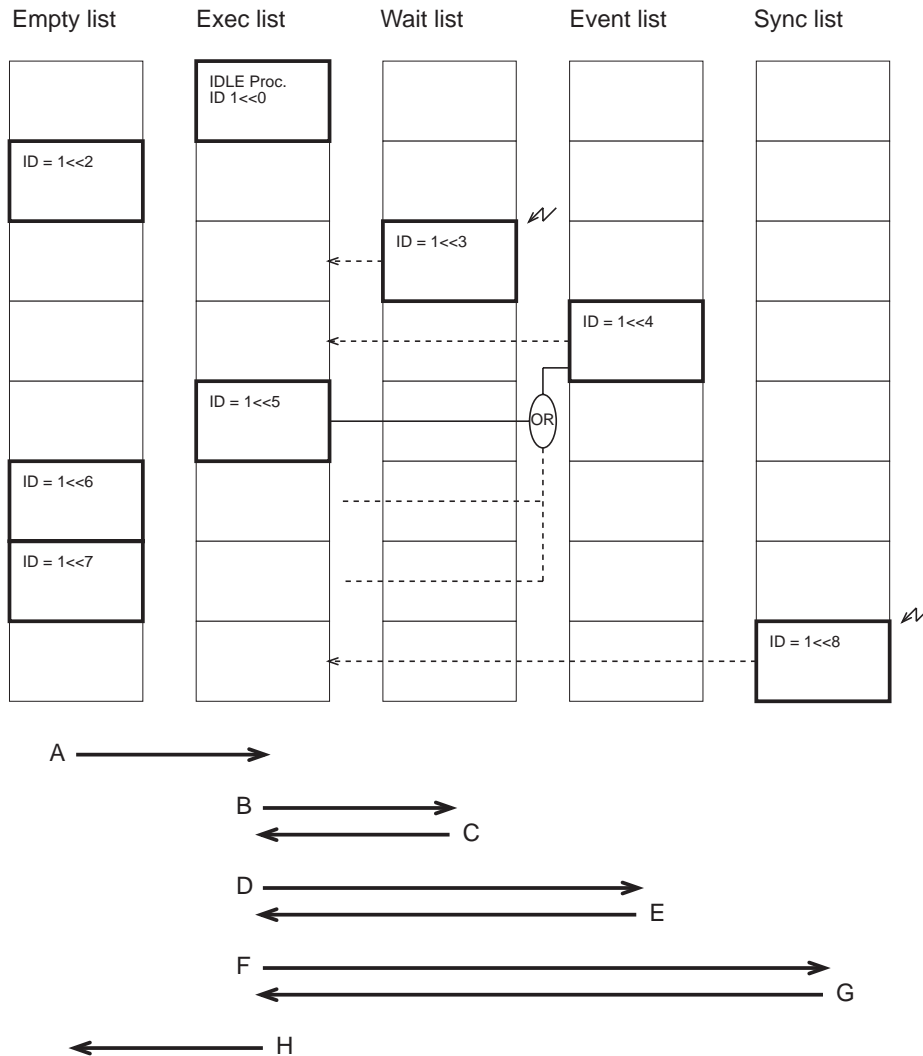


Generalities

This module manages the multitasking capabilities of the system. Thirty-two user tasks can be run simultaneously. The first task descriptor¹ on the execution list (IDLE task) is always present and is initialised during the start-up; the user cannot operate with this task. The basic functions necessary to operate with a multitasking kernel are implemented (task synchronisations, suspend tasks, global services, etc.). Time sharing allows to switch the tasks after 5 ms. The management of the context change is very fast; it takes only 1.5% of the CPU time. Figure 2 shows the main states of the task descriptors; here is the way they work:

- The empty list has to be considered as a tank of usable task descriptors. As many as thirty-two task descriptors can be used from this list.
- The execution list contains all the task descriptors that can run at a given time. This is the normal state for a task.
- The wait list contains all the task descriptors that have been suspended for a programmed time. This list is under the control of the real time clock PIT (1 ms of resolution). When a timeout occurs for a task, its descriptors will be placed again in the execution list.
- The event list contains all the task descriptors which are waiting for a software external event. The events have to be generated by other task. When an event occurs, its descriptor will be placed again in the execution list.
- The sync list contains all the task descriptors that are waiting for a hardware external synchronisation. The synchronisations are generated by hardware low level functions. When a synchronisation occurs, its descriptor will be placed again in the execution list.

1. A task descriptor is a structured memory representation of a task (or a process) on which the micro-kernel operates.



A: `tim_new_inst_task()`
Place a task descriptor into the "Execution list"

B: `tim_suspend_task()`
Suspend a task for a time; the task descriptor is moved into the "Wait list"

C: Timeout
Place a task descriptor into the "Execution list"

D: `tim_wait_event()`
Suspend a task for an event; the task descriptor is moved into the "Event list"

E: Event
Place a task descriptor into the "Execution list"

F: `tim_wait_sync()`
Suspend a task for an external event; the task descriptor is moved into the "Sync list"

G: External event
Place a task descriptor into the "Execution list"

H: `tim_remove_task()`
Place a task descriptor into the "Empty list"

Figure 2: Possible states of a task descriptor



`tim_reset()`

Init of the resources of the module. DO NOT USE FOR APPLICATIONS!

This system call inits the manager.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS tim_reset,0    ; execute the function
```

```
...  
tim_reset();
```



`tim_new_inst_task(textId, stack, procedure)`

Place a new task descriptor in the execution list. NOT USABLE FOR C APPLICATIONS!
Instead use `install_task(textId, stackLength, procedure)`.

This system call places one task descriptor in the execution list. As many as thirty-two task descriptors can be contained in the execution list. An error is returned if the execution list is full before the system call.

For C applications the user should use “`install_task(textId, stackLength, procedure)`”. The minimum stack length should be 800 (800 long words). However, the user can increase it according with the number of local variables that his program uses.

Input (stacking order):

<code>procedure</code>	Pointer on the task code.
<code>stack</code>	Pointer on the stack.
<code>textId</code>	Pointer on a string terminated with null ‘\0’.

Output:

<code>D0</code>	<code>2**0..2**31</code>	ID of the task descriptor.
<code>D0</code>	<code>-1</code>	Too many task descriptors in the execution list.

Call examples in assembler and C:

```
...
push.32 #R16^procedure ; pointer on the task code
push.32 #R16^stack ; pointer on the stack
push.32 #R16^textId ; pointer on an ASCII text
CALL_BIOS tim_new_inst_task,3 ; execute the function
test.32 D0 ;
jump,mi R8^Error ; too many task descriptors
move.32 D0, {A6}+id ; id of the task descriptor
```

```
void procedure(void)
{
...
}
```

```
int32 status;
uint32 id;
static char textId[] = "My task\n\r"
...
status = install_task(textId, 800, procedure);
if (status < 0) exit(0); /* Error, ... */
id = (uint32)status;
```



`tim_remove_inst_task(id)`

Remove a task descriptor. NOT USABLE FOR C APPLICATIONS! Instead use `kill_task(id)`.

This system call removes one task descriptor. An error is returned if the task descriptor does not exist. For C applications the user should use “`kill_task(id)`”.

Input (stacking order):

`id` ID of the task descriptor.

Output:

`D0` `2**0..2**31` ID of the task descriptor.

`D0` `-1` The task descriptor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+id          ; ID of the task descriptor
CALL_BIOS tim_remove_inst_task,1 ; execute the function
test.32  D0                ;
jump,mi  R8^Error         ; the task descriptor does not exist
move.32  D0, {A6}+id      ; id of the removed task descriptor

int32    status;
uint32   id;
...
status = kill_task(id);
if (status < 0) exit(0); /* Error, ... */

```




tim_generate_event()

Generate an event.

This system call generates an event used to synchronise other tasks. If a task was expecting a particular event (suspended inside the event list), it will be placed again in the execution list when the event occurs.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS tim_generate_event,0 ; execute the function  
  
...  
tim_generate_event();
```



tim_wait_event(taskMask)

Wait for an event.

This system call waits for an event generated by one or more other tasks. After this system call the current task descriptor is placed in the suspended event list. Only an event generated by another task can place the suspended task descriptor in the execution list.

Input (stacking order):

taskMask	Coding mask of the events which have to synchronise this task. The mask is usually the logical OR between the ID numbers of the task descriptors concerned.
----------	---

Output:

-

Call examples in assembler and C:

```
...
move.32  {A6}+id1,D0      ;
or.32    {A6}+id13,D0    ; IDs task descriptors 1 and 13
push.32  D0              ; wait for the task 1 or 13
CALL_BIOS tim_wait_event,1 ; execute the function

uint32   id1, id13;
...
tim_wait_event(id1|id13);
```



tim_get_id()

Return the task descriptor ID of the current task.

This system call returns the task descriptor ID number of the current task.

Input (stacking order):

-

Output:

id ID of the current task descriptor.

Call examples in assembler and C:

```
...
CALL_BIOS tim_get_id,0                ; execute the function
move.32    D0, {A6}+id                ; id of the current task descriptor

uint32    id;
...
id = tim_get_id();
```



tim_get_ticcount()

Return the number of tic count from the system call "tim_start_kernel".

This system call returns the number of tic count from the system call "tim_start_kernel". The value is expressed in milliseconds.

Input (stacking order):

-

Output:

ticCount value of the tic count.

Call examples in assembler and C:

```
...
CALL_BIOS tim_get_ticcount,0 ; execute the function
move.32 D0,{A6}+ticCount ; the value

uint32 ticCount;
...
ticCount = tim_get_ticcount();
```



tim_run_kernel()

Start the execution of the scheduled tasks. DO NOT USE FOR APPLICATIONS!

This system call starts the execution of the kernel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS tim_run_kernel,0 ; execute the function
```

```
...  
tim_run_kernel();
```



`tim_switch_fast()`

Stop the current task and switch to another one.

This system call stops immediately the execution of the current task and switches to another one. If only one task descriptor is inside the execution list, the switched task will be rescheduled immediately.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS tim_switch_fast,0 ; execute the function  
  
...  
tim_switch_fast();
```



`tim_lock()`

Lock the time sharing.

This system call locks the time sharing. Only the current task is executed. The system call is useful to protect critical resources (memory structures or I/O accesses).

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS tim_lock,0          ; execute the function  
  
...  
tim_lock();
```




tim_unlock()

Unlock the time sharing.

This system call unlocks the time sharing. If more than one “tim_lock()” system call was executed, the same number of “tim_unlock()” system calls has to be performed.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS tim_unlock,0      ; execute the function  
  
...  
tim_unlock();
```



tim_define_association(reference, general)

Define an association between a string and a general pointer.

This system call makes it possible to define an association between a 16 character string and a general 32-bit pointer. This allows a task to export high level references. For example, task 1 manages everything about the sensors; it can publish a pointer on a sensor table with a high level reference. All the other tasks that need to work with sensors can obtain the sensor pointer via the global high level reference. The general table can contain 32 associations. An error is returned if the association table is full before the system call.

Input (stacking order):

general	General pointer.
reference	Pointer on a 16 char (max.) string terminated with null '\0'.

Output:

D0	0	Association created.
D0	-1	Association table full.

Call examples in assembler and C:

```

...
push.32  #{A6}+general          ; general pointer
push.32  #R16^reference         ; pointer on the reference
CALL_BIOS tim_define_association,2 ; execute the function
test.32  D0                     ;
jump,mi  R8^Error               ; too many associations

int32    status;
static  char reference[] = "Sensors";
uint32   *general;
...
status = tim_define_association(reference, general);
if (status < 0) return -1;

```



tim_find_association(reference)

Look for an association.

This system call allows to get a general pointer referenced with a 16 character string. The string has to match the information in the association table. An error is returned if there is no association.

Input (stacking order):

reference	Pointer on a 16 char (max.) string terminated with null '\0'.
-----------	--

Output:

D0	<> -1	General pointer.
D0	-1	No association.

Call examples in assembler and C:

```

...
push.32  #R16^reference          ; pointer on the reference
CALL_BIOS tim_find_association,1 ; execute the function
test.32  D0                      ;
jump,mi  R8^Error                ; no association
move.32  D0,A0                   ; general pointer

int32    status;
uint32   *general;
static   char reference[] = "Sensors";
...
status = tim_find_association(reference);
if (status < 0) return -1;
general = (uint32 *)status;

```



tim_remove_association(reference)

Remove an association from the table.

This system call removes an association from the table. An error is returned if there is no such association

Input (stacking order):

reference	Pointer on a 16 char (max.) string terminated with null \0.
-----------	--

Output:

D0	0	Association removed.
D0	-1	No association.

Call examples in assembler and C:

```
...
push.32  #R16^reference      ; pointer on the reference
CALL_BIOS tim_remove_association,1 ; execute the function
test.32  D0                  ;
jump,mi  R8^Error           ; no association

int32    status;
char     *reference;
...
status = tim_remove_association(reference);
if (status < 0) return -1;
```



`tim_wait_sync(syncMask)`

Wait for an external synchronisation.

This system call waits for an external synchronisation generated by some low-level actions. After this system call the current task descriptor is placed in the suspended sync list. Only an event generated by a low-level action can place the suspended task descriptor in the execution list.

Input (stacking order):

<code>syncMask= 2**0</code>	PID sample.
<code>syncMask= 2**1</code>	Trajectory terminated, on target.
<code>syncMask= 2**2</code>	Message sent by MSG manager.
<code>syncMask= 2**3</code>	Message received by MSG manager.
<code>syncMask= 2**4</code>	Message sent by SER manager.
<code>syncMask= 2**5</code>	Byte received by SER manager.
<code>syncMask= 2**6</code>	IR sensors sync (each sensors generate a sync).
<code>syncMask= 2**7</code>	IR sensors sync (the sensor 0 generates a sync).
<code>syncMask= 2**8</code>	IRQ interruption.

Output:

-

Call examples in assembler and C:

```

...
push.32    {A6}+syncMask        ; wait for ...
CALL_BIOS tim_wait_sync,1       ; execute the function

uint32     syncMask;
...
tim_wait_sync(syncMask);

```



tim_get_task_des_ptr()

Return the pointer on the main task descriptor.

This system call returns the pointer on the main task descriptor structure.

Input (stacking order):

-

Output:

taskDescriptor main task descriptor pointer.

Call examples in assembler and C:

```
...
CALL_BIOS tim_get_task_des_ptr,0 ; execute the function
move.32 D0, {A6}+taskDescriptor ; the value

PROCDESC *taskDescriptor;
...
taskDescriptor = tim_get_task_des_ptr();
```



K-Team

<http://www.k-team.com>

info@k-team.com

MOT

MOT manager (motion control)

Family ID: 'BIOS'

Table of content

<code>mot_reset()</code>	.48
<code>mot_config_speed_1m(motorNb, kp, ki, kd)</code>	.49
<code>mot_new_speed_1m(motorNb, speed)</code>	.50
<code>mot_get_position(motorNb)</code>	.51
<code>mot_get_speed(motorNb)</code>	.52
<code>mot_put_sensors_1m(motorNb, position)</code>	.53
<code>mot_stop()</code>	.54
<code>mot_new_position_1m(motorNb, position)</code>	.55
<code>mot_new_pwm_1m(motorNb, pwm)</code>	.56
<code>mot_new_speed_2m(speed1, speed0)</code>	.57
<code>mot_config_position_1m(motorNb, kp, ki, kd)</code>	.58
<code>mot_put_sensors_2m(position1, position0)</code>	.59
<code>mot_new_position_2m(position1, position0)</code>	.60
<code>mot_config_profil_1m(motorNb, maxSpeed, maxAcceleration)</code>	.61
<code>mot_get_status(motorNb)</code>	.62
<code>mot_new_pwm_2m(pwm1, pwm0)</code>	.63



Generalities

This module controls all the resources necessary for the movement management. A classical PID controller coupled with a trapezoidal speed generator is available and allows a good speed and position control.

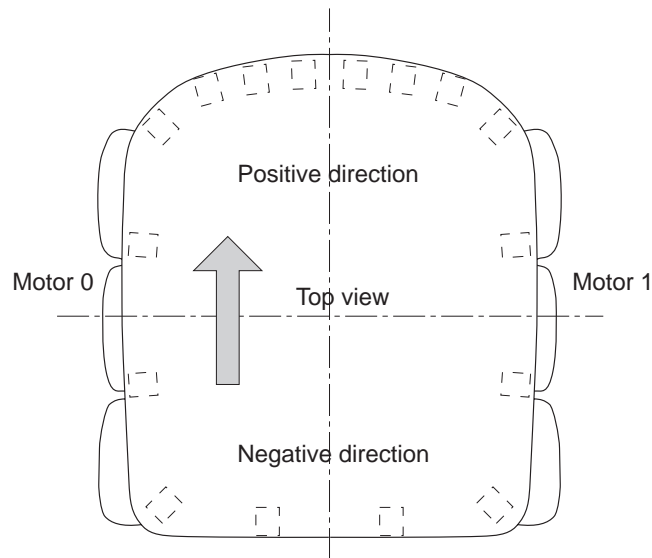


Figure 3: Motor number and direction position



The PID implementation

Speed and position controllers are based on a numerical PID implementation.

$$s(k) = u_p(k) + u_i(k) + u_d(k) \quad (\text{Eq. 1})$$

$$e(k) = \text{Reference} - \text{Measure}(k) \quad (\text{Eq. 2})$$

$$u_p(k) = K_p \cdot e(k) \quad (\text{Eq. 3})$$

$$u_i(k) = u_i(k-1) + K_i \cdot e(k) \quad (\text{Eq. 4})$$

$$u_d(k) = K_d[e(k) - e(k-1)] \quad (\text{Eq. 5})$$

- $s(k)$ is the output of controller. Its value is coded on 32-bits but only 16-bits are used. The “PWM” hardware implementation allows to use only 8-bits of the total dynamic. Here are the corresponding $PWM(k)$ and the direction command $d(k)$ values.

$$PWM(k) = \begin{cases} \frac{s(k)}{2^8} & \text{for } s(k) < (2^{15} - 1) \\ \text{not} \left[\frac{s(k)}{2^8} \right] & \text{for } s(k) \geq 2^{15} \end{cases} \quad (\text{Eq. 6})$$

$$d(k) = \text{sgn}[s(k)] \quad (\text{Eq. 7})$$

- $e(k)$ is the input error of the controller. Its value is coded on 32-bits but only 16-bits are used for the computation. If the absolute value of the error is greater than 2^{12} the cumulation sum u_i of the error of the integral part stops.
- $U_p(k)$ is the proportional contribution for the output of the controller. K_p is coded on 16-bits. The resultant product (Eq. 3) is coded on 32-bits.
- $U_i(k)$ is the integral contribution for the output of the controller. K_i is coded on 16-bits. The resultant product (Eq. 4) is coded on 32-bits.
- $U_d(k)$ is the integrate contribution for the output of the controller. K_d is coded on 16-bits. The resultant product (Eq. 5) is coded on 32-bits.

The speed profile generator

All the movements in the position control mode are under the control of the speed profile generator. For each sample, the speed generator computes the next desired speed. According to the maximum acceleration and speed, a trapezoidal speed profile (figure 4) is generated using the mechanical equations.

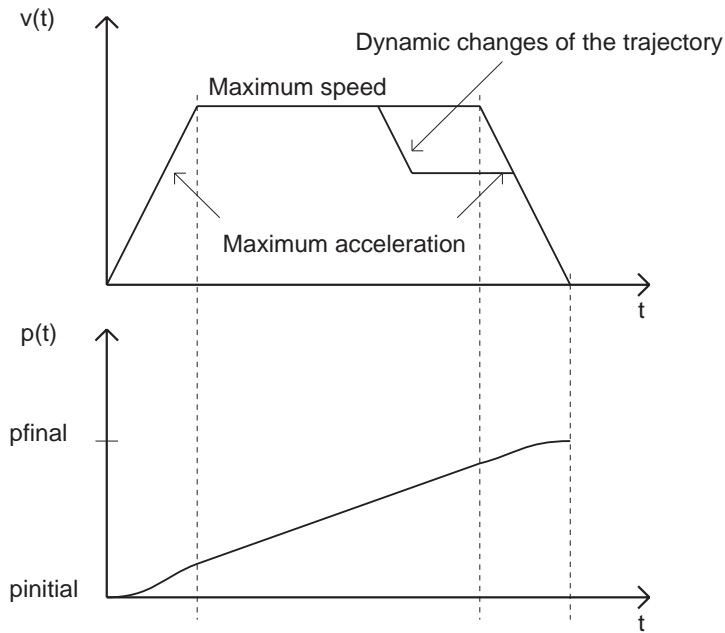


Figure 4: Speed and position movements



`mot_reset()`

Init the resources of the manager.

This system system call inits the manager. The PID is initialised in the speed control mode.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS mot_reset,0          ; execute the function
```

```
...  
mot_reset();
```



mot_config_speed_1m(motorNb, kp, ki, kd)

Initialise the speed PID coefficients for one motor.

This system call initialises the speed PID coefficients for one motor. As usual, the values stacked are coded on 32-bits, but only the LSW is used for the PID. The following good PID coefficients, lab tested, can be used for nearly all applications. An error is returned if the motor does not exist.

Proportional coefficient: $kp = 1000$

Integral coefficient: $ki = 800$

Derivative coefficient: $kd = 100$

Input (stacking order):

kd	Derivative coefficient.
ki	Integral coefficient.
kp	Proportional coefficient.
motorNb	Number of the motor [0..1].

Output:

D0	0	OK.
D0	-1	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+kd          ; derivative
push.32  {A6}+ki          ; integral
push.32  {A6}+kp          ; proportional
push.32  {A6}+motorNb     ; motor number
CALL_BIOS mot_config_speed_1m,4; execute the function
test.32  D0                ;
jump,mi  R8^Error         ; the motor does not exist

int32    status;
uint32   motorNb, kp, ki, kd;
...
status = mot_config_speed_1m(motorNb, kp, ki, kd);
if (stautus < 0) return -1;

```



mot_new_speed_1m(motorNb, speed)

New speed for one motor.

This system call changes the speed of one motor. An error is returned if the motor does not exist.

Maximum forward speed: speed = +127 (units)

Maximum backward speed: speed = -128 (units)

The unit of the speed is: 0.03 mm/10 ms

Input (stacking order):

speed		Speed value.
motorNb		Number of the motor [0..1].

Output:

D0	0	OK.
D0	-1	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+speed           ; speed value
push.32  {A6}+motorNb        ; motor number
CALL_BIOS mot_new_speed_1m,2 ; execute the function
test.32  D0                   ;
jump,mi  R8^Error            ; the motor does not exist

int32    status, speed;
uint32   motorNb;
...
status = mot_new_speed_1m(motorNb, speed);
if (status < 0) return -1;

```



mot_get_position(motorNb)

Get the absolute position of one motor.

This system call returns the absolute position (coming from the incremental sensor) of one motor.

The unit of the position is: 0.03 mm.

Input (stacking order):

motorNb Number of the motor [0..11].

Output:

D0 position Absolute position value.

Call examples in assembler and C:

```
...
push.32  {A6}+motorNb      ; motor number
CALL_BIOS mot_get_position,1 ; execute the function
move.32  D0, {A6}+position  ; position value

int32    position;
uint32   motorNb;
...
position = mot_get_position(motorNb);
```



mot_get_speed(motorNb)

Get the speed of one motor.

This system call returns the speed (difference of two absolute positions in one sample time) of one motor.

The unit of the speed is: 0.03 mm/10 ms.

Input (stacking order):

motorNb Number of the motor [0..1].

Output:

D0 speed Instantaneous speed.

Call examples in assembler and C:

```
...
push.32  {A6}+motorNb      ; motor number
CALL_BIOS mot_get_speed,1  ; execute the function
move.32  D0, {A6}+speed    ; speed value

int32    speed;
uint32   motorNb;
...
speed = mot_get_speed(motorNb);
```



mot_put_sensors_1m(motorNb, position)

New absolute position for one motor.

This system call sets the value of the incremental sensor of one motor. An error is returned if the motor does not exist.

The unit of the position: 0.03 mm.

Input (stacking order):

position	Position value.
motorNb	Number of the motor [0..1].

Output:

D0	0	OK.
D0	-1	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+position      ; position value
push.32  {A6}+motorNb      ; motor number
CALL_BIOS mot_put_sensors_1m,2 ; execute the function
test.32  D0                 ;
jump,mi  R8^Error          ; the motor does not exist

int32    status, position;
uint32   motorNb;
...
status = mot_put_sensors_1m(motorNb, position);
if (status < 0) return -1;

```




mot_stop()

Stop the motors and set PID coefficients to zero.

This system call immediately stops the two motors and sets the PID coefficients to zero.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS mot_stop,0           ; execute the function
```

```
...  
mot_stop();
```



mot_new_position_1m(motorNb, position)

New position for one motor.

This system call changes the position of one motor. An error is returned if the motor does not exist.

Maximum forward position:position = $+2^{31}-1$ (units)

Maximum backward position:position = -2^{31} (units)

The unit of the position is: 0.03 mm

Input (stacking order):

position	Position value.
motorNb	Number of the motor [0..1].

Output:

D0	0	OK.
D0	-1	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+position      ; position value
push.32  {A6}+motorNb      ; motor number
CALL_BIOS mot_new_position_1m,2; execute the function
test.32  D0                 ;
jump,mi  R8^Error          ; the motor does not exist

int32    status, position;
uint32   motorNb;
...
status = mot_new_position_1m(motorNb, position);
if (stautus < 0) return -1;

```



mot_new_pwm_1m(motorNb, pwm)

New pwm for one motor.

This system call changes the pwm of one motor. An error is returned if the motor does not exist.

Maximum forward pwm: pwm = $+2^7-1$ (units)

Maximum backward pwm: pwm = -2^7 (units)

Input (stacking order):

pwm		Pwm value.
motorNb		Number of the motor [0..1].

Output:

D0	0	OK.
D0	-1	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {R6}+pwm           ; pwm value
push.32  {R6}+motorNb       ; motor number
CALL_BIOS mot_new_pwm_1m,2  ; execute the function
test.32  D0                  ;
jump,mi  R8^Error           ; the motor does not exist

int32    status, pwm;
uint32   motorNb;
...
status = mot_new_pwm_1m(motorNb, pwm);
if (stautus < 0) return -1;

```



mot_new_speed_2m(speed1, speed0)

New speed for the two motors.

This system call changes the speed of the motors.

Maximum forward speed: speed = +127 (units)

Maximum backward speed: speed = -128 (units)

The unit of the speed is: 0.03 mm/10 ms

Input (stacking order):

speed0 Speed value.

speed1 Speed value.

Output:

-

Call examples in assembler and C:

```
...
push.32  {A6}+speed0              ; motor 0 speed value
push.32  {A6}+speed1              ; motor 1 speed value
CALL_BIOS mot_new_speed_2m,2      ; execute the function

int32     speed0, speed1;
...
mot_new_speed_2m(speed1, speed0);
```



`mot_config_position_1m(motorNb, kp, ki, kd)`

Initialise the position PID coefficients for one motor.

This system call initialises the position PID coefficients for one motor. As usual, the values stacked are coded on 32-bits, but only the LSW is used for the PID. The following good PID coefficients, lab tested, can be used for nearly all applications. An error is returned if the motor does not exist.

Proportional coefficient: `kp = 400`

Integral coefficient: `ki = 4`

Derivative coefficient: `kd = 400`

Input (stacking order):

<code>kd</code>	Derivative coefficient.
<code>ki</code>	Integral coefficient.
<code>kp</code>	Proportional coefficient.
<code>motorNb</code>	Number of the motor [0..1].

Output:

<code>D0 = 0</code>	OK.
<code>D0 = -1</code>	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+kd          ; derivative
push.32  {A6}+ki          ; integral
push.32  {A6}+kp          ; proportional
push.32  {A6}+motorNb     ; motor number
CALL_BIOS mot_config_position_1m,4 ; execute the function
test.32  D0                ;
jump,mi  R8^Error         ; the motor does not exist

int32    status;
uint32   motorNb, kp, ki, kd;
...
status = mot_config_position_1m(motorNb, kp, ki, kd);
if (status < 0) return -1;

```



mot_put_sensors_2m(position1, position0)

New absolute position for the two motors.

This system call changes the value of one incremental sensor of the two motors.

The unit of the position: 0.03 mm.

Input (stacking order):

position0	Position of the sensor 0.
position1	Position of the sensor 1.

Output:

-

Call examples in assembler and C:

```
...
push.32 {A6}+position0 ; position 0
push.32 {A6}+position1 ; position 1
CALL_BIOS mot_put_sensors_2m,2 ; execute the function

int32 position0, position1;
...
mot_put_sensors_2m(position1, position0);
```



mot_new_position_2m(position1, position0)

New position for the two motors.

This system call changes the position of the two motors.

Maximum forward position: position = $+2^{31}-1$ (units)

Maximum backward position: position = -2^{31} (units)

The unit of the position is: 0.03 mm

Input (stacking order):

position0 Position of the motor 0.

position1 Position of the motor 1.

Output:

-

Call examples in assembler and C:

```
...
push.32  {A6}+position0      ; position 0
push.32  {A6}+position1      ; position 1
CALL_BIOS mot_new_position_2m,2; execute the function

int32    position0, position1;
...
mot_new_position_2m(position1, position0);
```



mot_config_profil_1m(motorNb, maxSpeed, maxAcceleration)

Set-up the coefficients of the profile controller for one motor.

This system call initialises the speed and the acceleration coefficients for the profile controller for one motor. As usual, the values stacked are coded on 32-bits. The acceleration representation is coded in fixed point 24.8 (24-bits integer and 8-bits fractionary). The following good coefficients, lab tested, can be used for nearly all applications. An error is returned if the motor does not exist.

Maximum of the speed: maxSpeed = 20 (units)

Maximum of the acceleration: maxAcceleration = 0.25 (units)

The unit of the speed is: 0.03 mm/10 ms

The unit of the acceleration is: 0.03 mm/(10 ms ^ 2)

Input (stacking order):

maxAcceleration	Acceleration coefficient.
maxSpeed	Speed coefficient.
motorNb	Number of the motor [0..1].

Output:

D0	0	OK.
D0	-1	The motor does not exist.

Call examples in assembler and C:

```

...
push.32  {A6}+maxAcceleration    ; acceleration
push.32  {A6}+maxSpeed          ; speed
push.32  {A6}+motorNb           ; motor number
CALL_BIOS mot_config_profil_1m,3 ; execute the function
test.32  D0                      ;
jump,mi  R8^Error                ; the motor does not exist

int32    status;
uint32   motorNb, maxAcceleration, maxSpeed;
...
status = mot_config_profil_1m(motorNb, maxSpeed, maxAcceleration);
if (stautus < 0) return -1;

```




mot_get_status(motorNb)

Get the status of the motion for one motor.

This system call gives the status of the motion controller (speed and position) for one motor.

Input (stacking order):

motorNb Number of the motor [0..1].

Output:

D0	status	Status of the motion.
	2**0..2**15	error of the controller.
	2**16..2**17	mode (0 = speed, 1 = position, 2 = PWM).
	2**18	on target if mode = position.

Call examples in assembler and C:

```

...
push.32   {A6}+motorNb      ; motor number
CALL_BIOS mot_get_status,1  ; execute the function
move.32   D0, {A6}+status   ; status of the motion

int32     status;
uint32    motorNb;
...
status = mot_get_status(motorNb);
if (status < 0) return -1;

```



mot_new_pwm_2m(pwm1, pwm0)

New pwm for the two motors.

This system call changes the pwm of the two motors.

Maximum forward pwm: pwm = $+2^7-1$ (units)

Maximum backward pwm: pwm = -2^7 (units)

Input (stacking order):

pwm0 Pwm of the motor 0.

pwm1 Pwm of the motor 1.

Output:

-

Call examples in assembler and C:

```

...
push.32  {A6}+pwm0           ; pwm 0
push.32  {A6}+pwm1           ; pwm 1
CALL_BIOS mot_new_pwm_2m,2   ; execute the function

int32    pwm0, pwm1;
...
mot_new_pwm_2m(pwm1, pwm0);

```



K-Team

<http://www.k-team.com>
info@k-team.com

SENS

SENS manager (IR sensors and analog manager)

Family ID: 'BIOS'

Table of content

<code>sens_reset()</code>66
<code>sens_get_reflected_value(sensorNb)</code>67
<code>sens_get_ambient_value(sensorNb)</code>68
<code>sens_get_pointer()</code>69
<code>sens_get_ana_value(inputNb)</code>70

Generalities

This module manages the eight IR sensors (figure 5) and all the analog inputs. The hardware which controls the different phases of the IR sensor measure is completely under the control of this manager.

- Channel 0: is used to get the value "pulse light" of the left IR sensors.
- Channel 1: is used to get the value "ambient light" of the left IR sensors.
- Channel 2: is used to get the value "pulse light" of the right IR sensors.
- Channel 3: is used to get the value "ambient light" of the right IR sensors.
- Channels 4 to 9: user.

The sequence necessary to manage the IR sensors is divided into four phases.

- Phase 0: read the ambient light value.
- Phase 1: with a first sample/hold stores the ambient light.
- Phase 2: turn on the LED of the selected sensor
- Phase 3: read the ambient light + LED light value.

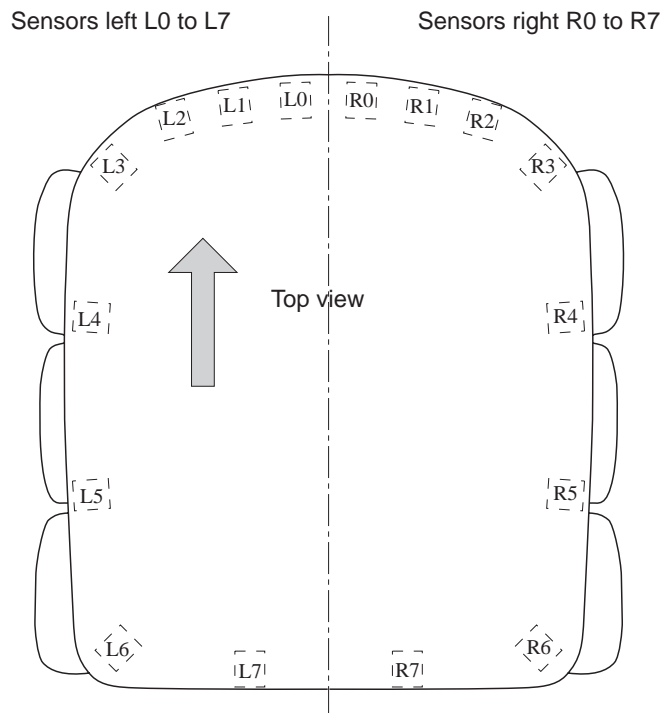


Figure 5: Position and number of the sensors. Left sensors take the numbers 0 to 7, right sensors from 8 to 15.



sens_reset()

Init of the resources of the manager.

This system call inits the manager.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS sens_reset,0      ; execute the function
```

```
...  
sens_reset();
```



`sens_get_reflected_value(sensorNb)`

Get the reflected value of one sensor.

This system call returns the reflected value of the sensor selected. The number 0 to 15 are used to select one sensor. An error is returned if the sensor does not exist.

Input (stacking order):

`sensorNb` Number of the sensor [0..15].

Output:

`D0` `sensorValue` Value of the sensor.

`D0` `-1` The sensor does not exist.

Call examples in assembler and C:

```

...
push.32   {A6}+sensorNb           ; sensor number
CALL_BIOS sens_get_reflected_value,1; execute the function
test.32   D0                       ;
jump,mi   R8^Error                ; the sensor does not exist
move.32   D0,{A6}+sensorValue     ; sensor value

int32     status;
uint32    sensorValue, sensorNb;

...
status = sens_get_reflected_value(sensorNb);
if (status < 0) return -1;
sensorValue = (uint32)status;

```



`sens_get_ambient_value(sensorNb)`

Get the ambient value of one sensor.

This system call returns the ambient value of the sensor selected. The number 0 to 15 are used to select one sensor. An error is returned if the sensor does not exist.

Input (stacking order):

`sensorNb` Number of the sensor [0..15].

Output:

`D0` `sensorValue` Value of the sensor.

`D0` -1 The sensor does not exist.

Call examples in assembler and C:

```

...
push.32   {A6}+sensorNb           ; sensor number
CALL_BIOS sens_get_ambient_value,1 ; execute the function
test.32   D0                       ;
jump,mi   R8^Error                 ; the sensor does not exist
move.32   D0, {A6}+sensorValue     ; sensor value

int32     status;
uint32    sensorValue, sensorNb;

...
status = sens_get_ambient_value(sensorNb);
if (status < 0) return -1;
sensorValue = (uint32)status;

```



`sens_get_pointer()`

Get the pointer on the "IRSENSOR" structure.

This system call returns the pointer on the "IRSENSOR" structure; pointer + 0 to 15 are used to point the value of the processed sensor (K x (Ambient - Pulsed)). Pointer + 16 to 31 are used to point the value of the ambient light. Here is the sensor structure:

```
#define    SENSORNB    16
#define    struct      {
                uint16    oProximitySensor [SENSORNB];
                uint16    oAmbientLightSensor [SENSORNB];
            } IRSENSOR;
```

Input (stacking order):

-

Output:

D0 sensor Pointer on the structure sensor.

Call examples in assembler and C:

```
...
CALL_BIOS    sens_get_pointer,0    ; execute the function
push.32     D0                    ;
pop.32      A1                    ; pointer on sensor structure

IRSENSOR    *sensor;
...
sensor = sens_get_pointer();
```




`sens_get_ana_value(inputNb)`

Get the value of one analog input.

This system call returns the value of the analog input selected. An error is returned if the input does not exist. Ten channels are used:

- 0: Reserved (left reflected light value)
- 1: Reserved (left ambient light value)
- 2: Reserved (right reflected light value)
- 3: Reserved (right ambient light value)
- 4: User
- 5: User
- 6: User
- 7: User
- 8: User
- 9: User

Input (stacking order):

`inputNb` Number of the analog input [0..9].

Output:

`D0` `analogValue` Analog value.

`D0` `-1` The input does not exist.

Call examples in assembler and C:

```
...
push.32   {A6}+inputNb           ; number of the input
CALL_BIOS sens_get_ana_value,1   ; execute the function
test.32   D0                     ;
jump,mi   R8^Error               ; the input does not exist
move.32   D0, {A6}+analogValue   ; analog value
```

```
int32     status;
uint32    analogValue, inputNb;
...
status = sens_get_ana_value(inputNb);
if (status < 0) return -1;
analogValue = (uint32)status;
```



MSG

Rev. 2.00

K-Team

<http://www.k-team.com>

info@k-team.com

MSG

MSG manager (local multi-microcontroller network manager)

Family ID: 'BIOS'

Table of content

<code>msg_reset()</code>	.73
<code>msg_reserve_channel(channelNb)</code>	.74
<code>msg_release_channel(channelNb)</code>	.75
<code>msg_send_message(msgS, sizeS)</code>	.76
<code>msg_receive_message(msgR, sizeR)</code>	.77
<code>msg_snd_rec_message(msgS, sizeS, msgR, sizeR, rep)</code>	.78

Generalities

This module manages all the communications between the main microcontroller of the system (master) and the different microcontrollers (slaves) available on the network. The small local network controlled by this module operates in a single master multi-slave configuration. The protocol is always supervised by the master microcontroller (star topology). Figure 6 shows how a message is coded and the different phases of the protocol.

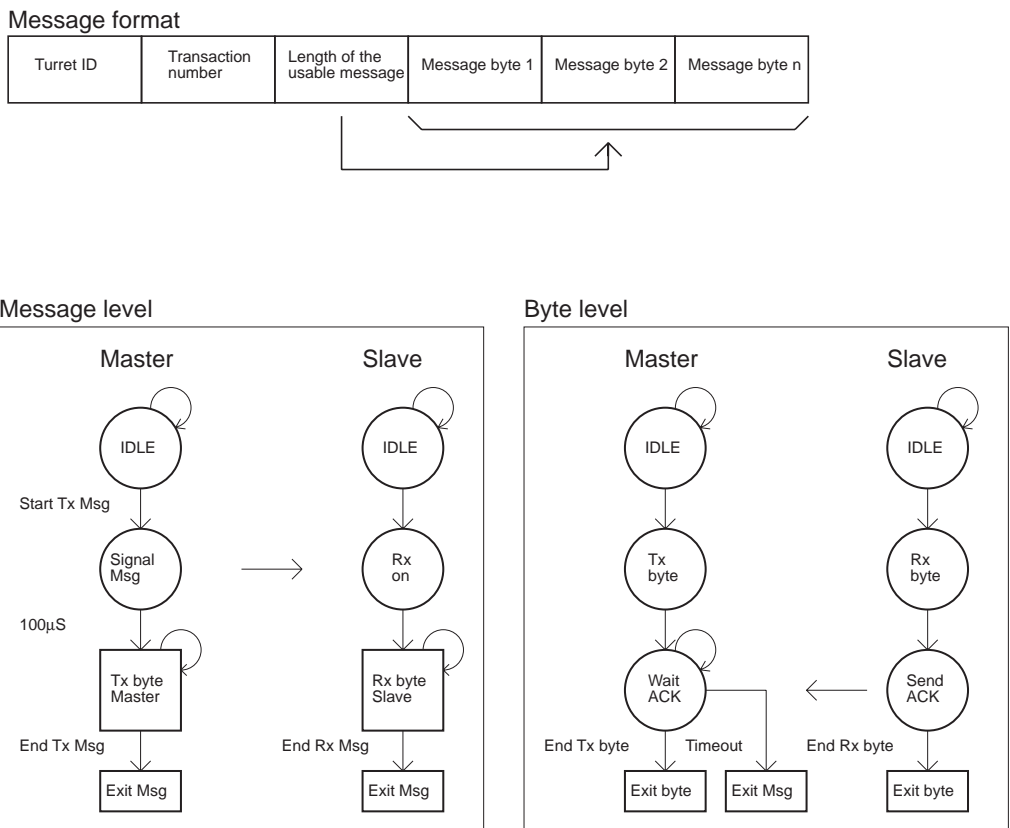


Figure 6: Message format and protocol sequences



`msg_reset()`

Init of the resources of the manager.

This system call inits the manager.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS msg_reset,0           ; execute the function
```

```
...  
msg_reset();
```



msg_reserve_channel(channelNb)

Reserve the logical channel of the network.

This system call reserves the logical channel of the network for a transaction. The network is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

channelNb Number of the logical channel.

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	The channel does not exist.
D0	-2	The channel busy.

Call examples in assembler and C:

```

push.32   {R6}+channelNb      ; channel number
CALL_BIOS msg_reserve_channel,1; execute the function
test.32   D0                  ;
jump,mi   R8^Error            ; wait ...

```

```

int32     status;
...
status = msg_reserve_channel(channelNb);
if (status < 0) return -1;

```



`msg_release_channel(channelNb)`

Release the logical channel of the network.

This system call releases the logical channel of the network. The other tasks can now use this channel.

Input (stacking order):

`channelNb` Number of the logical channel.

Output:

`D0` `0` Channel released.

`D0` `-1` The channel does not exist.

Call examples in assembler and C:

```
push.32    {A6}+channelNb        ; channel number  
CALL_BIOS msg_release_channel,1; execute the function
```

```
int32      status;  
...  
status = msg_release_channel(channelNb);  
if (status < 0) return -1;
```



`msg_send_message(messageS, sizeS)`

Send one message.

This system call sends one message on the network. The status of this call is given back. An error is returned if the message was not sent because of a time-out error.

Input (stacking order):

<code>sizeS</code>	Size of the message.
<code>messageS</code>	Pointer on the message.

Output:

<code>D0</code>	<code>0</code>	Message sent correctly.
<code>D0</code>	<code>-1</code>	Message not sent because of a time-out error.

Call examples in assembler and C:

```

...
push.32    {A6}+sizeS          ; size of the buffer
push.32    #{A6}+messageS     ; pointer on the message
CALL_BIOS  msg_send_message,2 ; execute the function
test.32    D0                  ;
jump,mi    R8^Error           ; time-out error

int32      status;
uint8      *messageS;
uint32     sizeS;
...
status = msg_send_message(messageS, sizeS);
if (stautus < 0) return -1;

```



`msg_receive_message(msgR, sizeR)`

Receive one message.

This system call waits for one message on the network. The status of this system call is given back. An error is returned if the message was not received because of a time-out error.

Input (stacking order):

<code>sizeR</code>	Size of the message.
<code>messageR</code>	Pointer on the message.

Output:

<code>D0</code>	<code>0</code>	Message received correctly.
<code>D0</code>	<code>-1</code>	Message not received because of a time-out error.

Call examples in assembler and C:

```

...
push.32  {A6}+sizeR      ; size of the buffer
push.32  #{A6}+messageR ; pointer on the message
CALL_BIOS msg_receive_message,2; execute the function
test.32  D0              ;
jump,mi  R8^Error       ; time-out error

int32    status;
uint8    *messageR;
uint32   sizeR;
...
status = msg_receive_message(messageR, sizeR);
if (stautus < 0) return -1;

```




`msg_snd_rec_message(msgS, sizeS, msgR, sizeR, rep)`

Send and receive one message.

This system call sends one message on the network and waits for an answer. An error is returned if the message was not received because of a time-out error.

Input (stacking order):

<code>rep</code>	Number of tries if error.
<code>sizeR</code>	Size of the message (to be received).
<code>msgR</code>	Pointer on the message (to be received).
<code>sizeS</code>	Size of the message (to be sent).
<code>msgS</code>	Pointer on the message (to be sent).

Output:

<code>D0</code>	<code>0</code>	Message sent and received correctly.
<code>D0</code>	<code>-1</code>	Message not sent nor received because of a time-out error (too many reps).

Call examples in assembler and C:

```

...
push.32 #2 ; try 2 times
push.32 {A6}+sizeR ; size of the buffer (R)
push.32 #{A6}+msgR ; pointer on the message (R)
push.32 {A6}+sizeS ; size of the buffer (S)
push.32 #{A6}+msgS ; pointer on the message (S)
CALL_BIOS msg_snd_rec_message,5; execute the function
test.32 D0 ;
jump,mi R8^Error ; time-out error

```

```

int32 status;
uint8 *msgS, *msgR;
uint32 sizeS, sizeR, rep;
...
status = msg_snd_rec_message(msgS, sizeS, msgR, sizeR, rep);
if (status < 0) return -1;

```



Franzi Edo.

K-Team S.A.

franzi@k-team.com

MMA

MMA manager (Multi Microcontroller Adapter manager)

Family ID: 'BIOS'

Table of content

mma_reset()	.81
mma_reserve_channel_0()	.82
mma_reserve_channel_1()	.83
mma_reserve_channel_2()	.84
mma_release_channel_0()	.85
mma_release_channel_1()	.86
mma_release_channel_2()	.87
mma_send_buffer_0(buffer, size)	.88
mma_send_buffer_1(buffer, size)	.89
mma_send_buffer_2(buffer, size)	.90
mma_receive_byte_0()	.91
mma_receive_byte_1()	.92
mma_receive_byte_2()	.93
mma_tx_status_0()	.94
mma_tx_status_1()	.95
mma_tx_status_2()	.96
mma_rx_status_0()	.97
mma_rx_status_1()	.98
mma_rx_status_2()	.99



Generalities

This module manages the communications via the multi-microcontroller parallel channel MMA (Multi-Micro-controller Adapter). All the operations are executed by interruptions. The interface with the MMA is achieved by using circular buffers; thus long waiting polling periods are avoided.



mma_reset()

Init of the resources of the manager.

This system call inits the manager.

Input:

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS mma_reset,0    ; execute the function
```

```
...  
mma_reset();
```



mma_reserve_channel_0()

Reserve the MMA 0 channel.

This system call reserves the MMA 0 channel for a transaction. The MMA channel is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

-

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	Channel busy.

Call examples in assembler and C:

```
...
CALL_BIOS mma_reserve_channel_0,0 ; execute the function
test.32 D0 ;
jump,mi R8^Error ; wait ...

int32 status;
...
status = mma_reserve_channel_0();
if (status < 0) return -1;
```



mma_reserve_channel_1()

Reserve the MMA 1 channel.

This system call reserves the MMA 1 channel for a transaction. The MMA channel is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

-

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	Channel busy.

Call examples in assembler and C:

```
...
CALL_BIOS mma_reserve_channel_1,0 ; execute the function
test.32 D0 ;
jump,mi R8^Error ; wait ...

int32 status;
...
status = mma_reserve_channel_1();
if (status < 0) return -1;
```



mma_reserve_channel_2()

Reserve the MMA 2 channel.

This system call reserves the MMA 2 channel for a transaction. The MMA channel is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

-

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	Channel busy.

Call examples in assembler and C:

```
...  
CALL_BIOS mma_reserve_channel_2,0 ; execute the function  
test.32 D0 ;  
jump,mi R8^Error ; wait ...  
  
int32 status;  
...  
status = mma_reserve_channel_2();  
if (status < 0) return -1;
```



`mma_release_channel_0()`

Release the MMA 0 channel.

This system call releases the MMA 0 channel. The other tasks can now use this channel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS mma_release_channel_0,0 ; execute the function  
  
...  
mma_release_channel_0();
```




`mma_release_channel_1()`

Release the MMA 1 channel.

This system call releases the MMA 1 channel. The other tasks can now use this channel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS mma_release_channel_1,0 ; execute the function  
  
...  
mma_release_channel_1();
```



`mma_release_channel_2()`

Release the MMA 2 channel.

This system call releases the MMA 2 channel. The other tasks can now use this channel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS mma_release_channel_2,0 ; execute the function  
  
...  
mma_release_channel_2();
```



mma_send_buffer_0(buffer, size)

Send one buffer by the MMA 0 channel.

This system call sends one buffer of less than 1024 bytes by the MMA 0 channel. An error is returned (if any).

Input (stacking order):

size	Size of the buffer to send.
buffer	Pointer on the buffer.

Output:

D0	0	OK.
D0	-1	Channel busy.
D0	-2	Size of the buffer excessive.
D0	-3	Size of the buffer = 0.

Call examples in assembler and C:

```

...
push.32    {A6}+size           ; size of the buffer to send
push.32    #{A6}+buffer       ; pointer on the buffer
CALL_BIOS  mma_send_buffer_0,2 ; execute the function
test.32    D0                  ;
jump,mi    R8^Error           ; channel error

int32      status;
uint8      *buffer;
uint32     size;
...
status = mma_send_buffer_0(buffer, size);
if (status < 0) return status;

```



mma_send_buffer_1(buffer, size)

Send one buffer by the MMA 1 channel.

This system call sends one buffer of less than 1024 bytes by the MMA 1 channel. An error is returned (if any).

Input (stacking order):

size	Size of the buffer to send.
buffer	Pointer on the buffer.

Output:

D0	0	OK.
D0	-1	Channel busy.
D0	-2	Size of the buffer excessive.
D0	-3	Size of the buffer = 0.

Call examples in assembler and C:

```

...
push.32    {A6}+size           ; size of the buffer to send
push.32    #{A6}+buffer       ; pointer on the buffer
CALL_BIOS  mma_send_buffer_1,2 ; execute the function
test.32    D0                  ;
jump,mi    R8^Error           ; channel error

int32      status;
uint8      *buffer;
uint32     size;
...
status = mma_send_buffer_1(buffer, size);
if (status < 0) return status;

```



mma_send_buffer_2(buffer, size)

Send one buffer by the MMA 2 channel.

This system call sends one buffer of less than 1024 bytes by the MMA 2 channel. An error is returned (if any).

Input (stacking order):

size	Size of the buffer to send.
buffer	Pointer on the buffer.

Output:

D0	0	OK.
D0	-1	Channel busy.
D0	-2	Size of the buffer excessive.
D0	-3	Size of the buffer = 0.

Call examples in assembler and C:

```

...
push.32    {A6}+size           ; size of the buffer to send
push.32    #{A6}+buffer       ; pointer on the buffer
CALL_BIOS  mma_send_buffer_2,2 ; execute the function
test.32    D0                  ;
jump,mi    R8^Error           ; channel error

int32      status;
uint8      *buffer;
uint32     size;
...
status = mma_send_buffer_2(buffer, size);
if (status < 0) return status;

```



mma_receive_byte_0()

Receive one byte by the MMA 0 channel.

This system call looks for the reception buffer of the MMA 0 channel if one byte is available.

Input (stacking order):

-

Output:

D0	+16'000000nn	nn = byte.
D0	-1	Buffer empty.

Call examples in assembler and C:

```

...
CALL_BIOS mma_receive_byte_0,0 ; execute the function
test.32   D0                    ;
jump,mi   R8^Error              ; channel error
move.8    D0,{A6}+aByte         ; a character

int32     status;
uint8     aByte;

...
status = mma_receive_byte_0();
if (status < 0) return -1;
aByte = (uint8)status;

```



mma_receive_byte_1()

Receive one byte by the MMA 1 channel.

This system call looks for the reception buffer of the MMA 1 channel if one byte is available.

Input (stacking order):

-

Output:

D0	+16'000000nn	nn = byte.
D0	-1	Buffer empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_receive_byte_1,0 ; execute the function
test.32   D0                    ;
jump,mi   R8^Error              ; channel error
move.8    D0,{A6}+aByte         ; a character

int32     status;
uint8     aByte;

...
status = mma_receive_byte_1();
if (status < 0) return -1;
aByte = (uint8)status;
```



mma_receive_byte_2()

Receive one byte by the MMA 2 channel.

This system call looks for the reception buffer of the MMA 2 channel if one byte is available.

Input (stacking order):

-

Output:

D0	+16'000000nn	nn = byte.
D0	-1	Buffer empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_receive_byte_2,0 ; execute the function
test.32   D0                    ;
jump,mi   R8^Error              ; channel error
move.8    D0,{A6}+aByte         ; a character

int32     status;
uint8     aByte;

...
status = mma_receive_byte_2();
if (status < 0) return -1;
aByte = (uint8)status;
```




mma_tx_status_0()

Get the status of the MMA 0 channel transmitter.

This system call looks for the status of the MMA 0 channel transmitter.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_tx_status_0,0      ; execute the function
test.32  D0                      ;
jump,mi  R8^Error                ; buffer not empty

int32    status;
...
status = mma_tx_channel_0();
if (status < 0) return -1;
```



mma_tx_status_1()

Get the status of the MMA 1 channel transmitter.

This system call looks for the status of the MMA 1 channel transmitter.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...  
CALL_BIOS mma_tx_status_1,0      ; execute the function  
test.32  D0                      ;  
jump,mi  R8^Error                ; buffer not empty  
  
int32    status;  
...  
status = mma_tx_channel_1();  
if (status < 0) return -1;
```



mma_tx_status_2()

Get the status of the MMA 2 channel transmitter.

This system call looks for the status of the MMA 2 channel transmitter.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_tx_status_2,0      ; execute the function
test.32  D0                      ;
jump,mi  R8^Error                ; buffer not empty

int32    status;
...
status = mma_tx_channel_2();
if (status < 0) return -1;
```



mma_rx_status_0()

Get the status of the MMA 0 channel receiver.

This system call looks for the status of the MMA 0 channel receiver.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_rx_status_0,0 ; execute the function
test.32 D0 ;
jump,mi R8^Error ; buffer not empty

int32 status;
...
status = mma_rx_channel_0();
if (status < 0) return -1;
```



mma_rx_status_1()

Get the status of the MMA 1 channel receiver.

This system call looks for the status of the MMA 1 channel receiver.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_rx_status_1,0    ; execute the function
test.32   D0                    ;
jump,mi   R8^Error             ; buffer not empty

int32     status;
...
status = mma_rx_channel_1();
if (status < 0) return -1;
```



mma_rx_status_2()

Get the status of the MMA 2 channel receiver.

This system call looks for the status of the MMA 2 channel receiver.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS mma_rx_status_2,0 ; execute the function
test.32 D0 ;
jump,mi R8^Error ; buffer not empty

int32 status;
...
status = mma_rx_channel_2();
if (status < 0) return -1;
```



SER

Rev. 1.00

K-Team

<http://www.k-team.com>
info@k-team.com

SER

SER manager (serial RS232 manager)

Family ID: 'BIOS'

Table of content

<code>ser_reset()</code>	102
<code>ser_reserve_channel()</code>	103
<code>ser_release_channel()</code>	104
<code>ser_config(baudrate)</code>	105
<code>ser_send_buffer(buffer, size)</code>	106
<code>ser_receive_byte()</code>	107
<code>ser_tx_status()</code>	108
<code>ser_rx_status()</code>	109



Generalities

This module manages the communications via the asynchronous serial channel SCI (Serial Communication Interface). All the operations are executed by interruptions. The interface with the SCI is achieved by using circular buffers; thus long waiting polling periods are avoided. The format used is fixed at 8-bits, 2-stop bits, no parity. Only the baudrate can be changed.



ser_reset()

Init of the resources of the manager.

This system call inits the manager. The baudrate is selected to 9600 bits/s.

Input:

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS ser_reset,0    ; execute the function
```

```
...  
ser_reset();
```



`ser_reserve_channel()`

Reserve the serial SCI channel.

This system call reserves the serial SCI channel for a transaction. The serial SCI channel is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

-

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	Channel busy.

Call examples in assembler and C:

```
...
CALL_BIOS ser_reserve_channel,0      ; execute the function
test.32  D0                          ;
jump,mi  R8^Error                    ; wait ...

int32    status;
...
status = ser_reserve_channel();
if (status < 0) return -1;
```



`ser_release_channel()`

Release the serial SCI channel.

This system call releases the serial SCI channel. The other tasks can now use this channel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS ser_release_channel,0 ; execute the function  
  
...  
ser_release_channel();
```



`ser_config(baudrate)`

Set-up the baudrate.

This system call allows to change the baudrate of the serial SCI channel; eight possibilities are available. An error is returned if the baudrate does not exist.

Input (stacking order):

baudrate	0->9600 B.
	1->600 B.
	2->1200 B.
	3->4800 B.
	4->9600 B.
	5->19200 B.
	6->38400 B.
	7->57600 B.
	8->115200 B.
	9->230400 B.

Output:

D0	0	OK.
D0	-1	The baudrate does not exist.

Call examples in assembler and C:

```

...
push.32    {R6}+baudrate      ; baudrate
CALL_BIOS ser_config, 1      ; execute the function
test.32    D0                 ;
jump,mi    R8^Error          ; the baudrate does not exist

int32      status;
uint32     baudrate;
...
status = ser_config(baudrate);
if (status < 0) return -1;

```



ser_send_buffer(buffer, size)

Send one buffer by the serial SCI channel.

This system call sends one buffer of less than 500 bytes by the serial SCI channel. It is under the control of the Tx interruption. An error is returned (if any).

Input (stacking order):

size		Size of the buffer to send.
buffer		Pointer on the buffer.

Output:

D0	0	OK.
D0	-1	Channel busy.
D0	-2	Size of the buffer excessive.
D0	-3	Size of the buffer = 0.

Call examples in assembler and C:

```

...
push.32    {A6}+size           ; size of the buffer to send
push.32    #{A6}+buffer       ; pointer on the buffer
CALL_BIOS  ser_send_buffer,2   ; execute the function
test.32    D0                  ;
jump,mi    R8^Error           ; channel error

int32      status;
uint8      *buffer;
uint32     size;
...
status = ser_send_buffer(buffer, size);
if (status < 0) return status;

```



`ser_receive_byte()`

Receive one byte by the serial SCI channel.

This system call looks for the reception buffer of the serial SCI channel if one byte is available. This system call is under control of the Rx interruption.

Input (stacking order):

-

Output:

D0 +16'000000nn nn = byte.

D0 -1 Buffer empty.

Call examples in assembler and C:

```
...
CALL_BIOS ser_receive_byte,0 ; execute the function
test.32 D0 ;
jump,mi R8^Error ; channel error
move.8 D0, {A6}+aByte ; a character

int32 status;
uint8 aByte;
...
status = ser_receive_byte();
if (status < 0) return -1;
aByte = (uint8)status;
```



`ser_tx_status()`

Get the status of the serial SCI channel transmitter.

This system call looks for the status of the serial SCI channel transmitter.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS ser_tx_status,0      ; execute the function
test.32  D0                    ;
jump,mi  R8^Error              ; buffer not empty

int32    status;
...
status = ser_tx_channel();
if (status < 0) return -1;
```



`ser_rx_status()`

Get the status of the serial SCI channel receiver.

This system call looks for the status of the serial SCI channel receiver.

Input (stacking order):

-

Output:

D0	0	Buffer empty.
D0	-1	Buffer not empty.

Call examples in assembler and C:

```
...
CALL_BIOS ser_rx_status,0      ; execute the function
test.32   D0                   ;
jump,mi   R8^Error            ; buffer not empty

int32     status;
...
status = ser_rx_channel();
if (status < 0) return -1;
```




VAR

Rev. 2.00

K-Team

<http://www.k-team.com>

info@k-team.com

VAR

VAR manager (misc. and interruption manager)

Family ID: 'BIOS'

Table of content

<code>var_reset()</code> 112
<code>var_get_jumper()</code> 113
<code>var_on_led(ledNb)</code> 114
<code>var_off_led(ledNb)</code> 115
<code>var_change_led(ledNb)</code> 116
<code>var_set_irq_vector(procedure)</code> 117
<code>var_enable_irq()</code> 118
<code>var_disable_irq()</code> 119
<code>var_set_exception_vector(procedure, exceptionNb)</code> 120
<code>var_cpu_speed(CPUSpeedValue)</code> 121
<code>var_get_extension(address)</code> 122
<code>var_put_extension(address, binaryValue)</code> 123



Generalities

This module manages different low level resources such as jumper reading, LEDs control and the user external interruption. The parallel extension bus of the system is also under the control of this module.

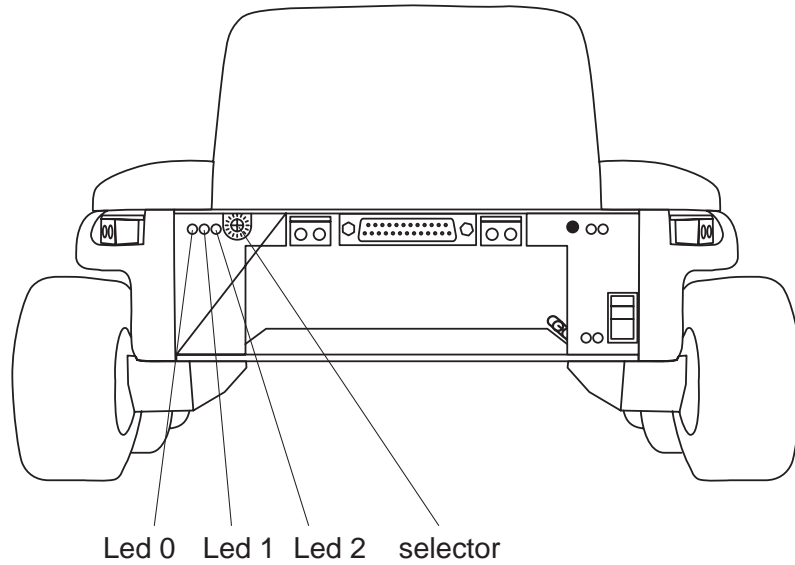


Figure 7: LED and jumper definitions



`var_reset()`

Init of the resources of the manager.

This system call inits the manager.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS var_reset,0      ; execute the function
```

```
...  
var_reset();
```



var_get_jumper()

Get the value of the selector.

This system call returns the state values of the selector of the main board.

Input (stacking order):

-

Output:

D0 jumperValue State of the jumpers.

Call examples in assembler and C:

```
...
CALL_BIOS var_get_jumper,0        ; execute the function
move.32    D0, {A6}+jumperValue ; jumper value

uint32     jumperValue;
...
jumperValue = var_get_jumper();
```



var_on_led(ledNb)

Turn on one LED.

This system call turns on a selected LED. An error is returned if the LED does not exist.

Input (stacking order):

ledNb Number of the LED.

Output:

D0 0 OK.

D0 -1 The LED does not exist.

Call examples in assembler and C:

```
...
push.32   {R6}+ledNb           ;
CALL_BIOS var_on_led,1         ; execute the function
test.32   D0                    ;
jump,mi   R8^Error             ; the LED does not exist
```

```
int32     status;
uint32    ledNb;
...
status = var_on_led(ledNb);
if (status < 0) return -1;
```



var_off_led(ledNb)

Turn off one LED.

This system call turns off a selected LED. An error is returned if the LED does not exist.

Input (stacking order):

ledNb Number of the LED.

Output:

D0 0 OK.

D0 -1 The LED does not exist.

Call examples in assembler and C:

```

...
push.32   {R6}+ledNb           ;
CALL_BIOS var_off_led,1       ; execute the function
test.32   D0                   ;
jump,mi   R8^Error            ; the LED does not exist

int32     status;
uint32    ledNb;
...
status = var_off_led(ledNb);
if (status < 0) return -1;

```




`var_set_irq_vector(procedure)`

Attribute one procedure to the user interruption.

This system call initialises the user interruption vector with the address of one procedure. The return code of the procedure has to be a “RTS” and not a “RTSF”.

Input (stacking order):

procedure Pointer on the procedure.

Output:

-

Call examples in assembler and C:

```
...
push.32   #R16^procedure      ;
CALL_BIOS var_set_irq_vector,1 ; execute the function
```

```
void      procedure(void)
{
    ...
}

...
var_set_irq_vector(procedure);
```




`var_enable_irq()`

Enable the user interruption.

This system call enables the user interruption.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS var_enable_irq,0 ; execute the function
```

```
...  
var_enable_irq();
```



`var_disable_irq()`

Disable the user interruption.

This system call disables the user interruption.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS var_disable_irq,0 ; execute the function
```

```
...  
var_disable_irq();
```



`var_set_exception_vector(procedure, exceptionNb)`

Attribute one procedure to an exception vector number.

This system call initialises an exception vector number with the address of one procedure. The return code of the procedure has to be "RTSF". An error is returned if the exception does not exist.

Input (stacking order):

<code>exceptionNb</code>	Number of the exception.
<code>procedure</code>	Pointer on the procedure.

Output:

<code>D0</code>	<code>0</code>	OK
<code>D0</code>	<code>-1</code>	The exception does not exist

Call examples in assembler and C:

```

...
push.32    {R6}+exceptionNb    ; exception number
push.32    #R16^procedure      ;
CALL_BIOS  var_set_exception,2  ; execute the function
test.32    D0                  ;
jump,mi    R8^Error            ; the exception does not exist

```

```

void      procedure(void)
{
    ...
}

```

```

int32     status;
uint32    exceptionNb;
...
status = var_set_exception(procedure, exceptionNb);
if (status < 0) return -1;

```



var_cpu_speed(CPUSpeedValue)

Change the speed of the CPU. !!! Avoid using this system call!!!

This system call changes the speed of the CPU. It has to be used very carefully; in fact, there are other devices which are influenced when the speed is changed (serial baudrate in particular). An error is returned if the speed does not exist.

Input (stacking order):

CPUSpeedValue	Speed number.
	0 = 16,78 MHz.
	1 = 8.38 MHz.

Output:

D0	0	OK.
D0	-1	The speed does not exist.

Call examples in assembler and C:

```

...
push.32    {A6}+CPUSpeedValue    ; the speed value
CALL_BIOS  var_cpu_speed,1       ; execute the function
test.32    D0                    ;
jump,mi    R8^Error              ; the speed does not exist

int32      status;
uint32     CPUSpeedValue;
...
status = var_cpu_speed(CPUSpeedValue);
if (status < 0) return -1;

```




`var_put_extension(address, binaryValue)`

Write on the extension bus.

This system call allows to write values on the extension bus of the system. The address is relative to the beginning of the memory-space [0..63].

Input (stacking order):

<code>binaryValue</code>	<code>0x000000bb</code> .
<code>address</code>	Relative address [0..63].

Output:

-

Call examples in assembler and C:

```
...
push.32  {A6}+binaryValue      ; the value
push.32  {A6}+address          ;
CALL_BIOS var_put_extension,2  ; execute the function

#define  ioport[10] = 0x10
uint32  binaryValue, *address;
...
address = (int32 *)ioport;
var_put_extension(address, binaryValue);
```



UIO

Rev. 1.00

K-Team

<http://www.k-team.com>
info@k-team.com

UIO

UIO manager (general INPUT / OUTPUT manager)

Family ID: 'BIOS'

Table of content

<code>uio_reset()</code>	126
<code>uio_on_out(outNbr)</code>	127
<code>uio_off_out(outNbr)</code>	128
<code>uio_change_out(outNbr)</code>	129
<code>uio_get_inputs()</code>	130
<code>uio_set_msk_irq_inputs(interruptMask)</code>	131
<code>uio_ack_irq_inputs(ackMask)</code>	132
<code>uio_get_status_irq_inputs()</code>	133
<code>uio_get_input_state(inputNbr)</code>	134

Generalities

This module manages the digital I/O port available on Koala. The analog channels are managed by the SENS manager.: channel 0 on the I/O port correspond to channel 4 of the A/D converter.

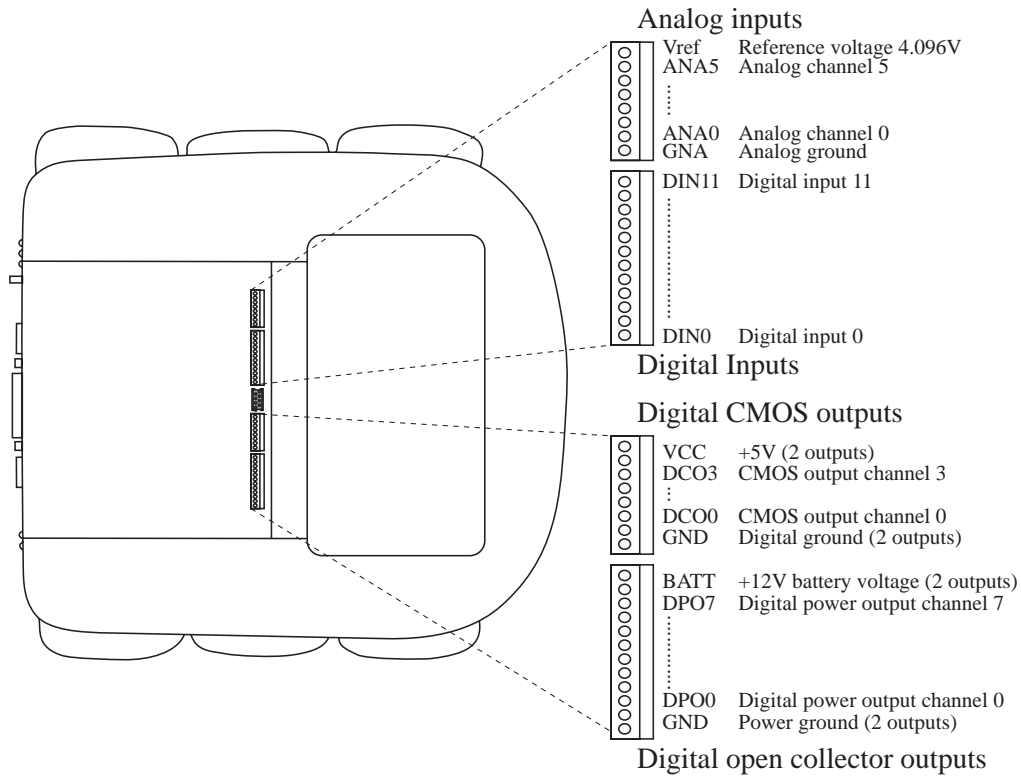


Figure 8: General I/O port definitions



`uio_reset()`

Init of the resources of the manager.

This system call inits the manager.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS uio_reset,0          ; execute the function
```

```
...  
uio_reset();
```




`uio_off_out(outNbr)`

Set OFF a digital output.

This system call switch OFF a digital output. Outputs number 0 to 8 correspond to the 8 digital power outputs. Outputs number 9 to 11 correspond to the 4 CMOS outputs.

Input (stacking order):

`outNbr` Output number

Output:

<code>D0</code>	<code>0</code>	OK.
<code>D0</code>	<code>-1</code>	The output does not exist.

Call examples in assembler and C:

```

...
push.32   {R6}+outNbr           ;
CALL_BIOS uio_off_out,1         ; execute the function
test.32   D0                     ;
jump,mi   R8^Error              ; the output does not exist

int32     status;
uint32    outNbr;
...
status = uio_off_out(outNbr);
if (status < 0) return -1;

```




uio_get_inputs()

Get the values of the digital inputs.

This system call returns the state values of the digital inputs. Bit 0 to 11 correspond to the state of inputs 0 to 11.

Input (stacking order):

-

Output:

D0 inputsValue State of the digital inputs.

Call examples in assembler and C:

```
...
CALL_BIOS uio_get_inputs,0        ; execute the function
move.32    D0,{A6}+inputsValue    ; general input value

uint32     inputsValue;
...
inputsValue = uio_get_inputs();
```



`uio_set_msk_irq_inputs(interruptMask)`

Set the interrupt mask on the general input port.

This system call sets the mask which enable or disable general input to become interrupt request lines. Bits 0 to 11 are the mask bits for inputs 0 to 11. Interrupt are generated if the bit is set to 1, are masked if set to 0.

Input (stacking order):

`interruptMask` mask enabling/disabling the interrupt

Output:

-

Call examples in assembler and C:

```
...
push.32  {A6}+interruptMask      ;
CALL_BIOS uio_set_msk_irq_inputs,1 ; execute the function

uint32   interruptMask;
...
uio_set_msk_irq_inputs(interruptMask);
```



`uio_ack_irq_inputs(ackMask)`

Acknowledge the interrupt on the general input port.

This system call acknowledge one or more interrupts coming from the general input lines. Bits 0 to 11 are the acknowledge bits for inputs 0 to 11. Writing a 0 to a bit acknowledge the corresponding interrupt and clear it.

Input (stacking order):

`ackMask` acknowledge mask clearing the interrupt

Output:

-

Call examples in assembler and C:

```
...
push.32  {A6}+ackMask          ;
CALL_BIOS uio_ack_irq_inputs,1  ; execute the function

uint32   ackMask;
...
uio_ack_irq_inputs(ackMask);
```



uio_get_status_irq_inputs()

Get the status of the interrupt inputs.

This system call returns the state values of the interrupts corresponding to the general inputs. Bit 0 to 11 correspond to the state of inputs 0 to 11.

Input (stacking order):

-

Output:

D0 interruptValue State of the interrupts.

Call examples in assembler and C:

```
...
CALL_BIOS uio_get_status_irq_inputs,0        ; execute the function
move.32    D0, {A6}+interruptValue        ; general input value

uint32     interruptValue;
...
interruptValue = uio_get_status_irq_inputs();
```




uio_get_input_state(inputNbr)

Get the value of one digital input.

This system call returns the state values of one digital input.

Input (stacking order):

inputNbr input number

Output:

D0 state OK.

D0 -1 The input does not exist.

Call examples in assembler and C:

```
...
push.32   {R6}+inputNbr      ;
CALL_BIOS uio_get_input_state,1; execute the function
test.32   D0                  ;
jump,mi   R8^Error           ; the input does not exist

int32     status;
uint32    inputNbr;
...
status = uio_get_input_state(inputNbr);
if (status < 0)
    return -1;
else
    return status;
```



CTR

Rev. 1.00

K-Team

<http://www.k-team.com>

info@k-team.com

CTR

CTR manager (control signals manager)

Family ID: 'BIOS'

Table of content

<code>ctr_reset()</code>	137
<code>ctr_get_ana_value(ctrChannel)</code>	138
<code>ctr_reserve_channel()</code>	139
<code>ctr_release_channel()</code>	140
<code>ctr_get_eeprom(address)</code>	141
<code>ctr_put_eeprom(address, binaryValue)</code>	142



Generalities

This module manages the control signals for the Koala robot, including the general power consumption, the motor power consumption and the management of specific devices related to this (battery eeprom, for instance).



`ctr_reset()`

Init of the resources of the manager.

This system call inits the manager.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS ctr_reset,0      ; execute the function
```

```
...  
ctr_reset();
```



ctr_get_ana_value(ctrChannel)

Get the value of a CTR analog channel.

This system call returns the value of the analog channel selected. An error is returned if the input does not exist. Six channels are used:

- 0: Battery voltage (measure unit: 20 mV).
- 1: General consumption current (measure unit: 8 mA).
- 2: Ambient temperature (measure unit: 0.1° C).
- 3: Left motor current (measure unit: 4 mA).
- 4: Right motor current (measure unit: 4 mA).
- 5: Battery temperature (measure unit: 0.1° C).

Input (stacking order):

ctrChannel Number of the analog CTR channel [0..5].

Output:

D0 analogValue Analog value.
D0 -1 The input does not exist.

Call examples in assembler and C:

```

...
push.32   {A6}+ctrChannel      ; number of the input
CALL_BIOS sens_get_ana_value,1 ; execute the function
test.32   D0                   ;
jump,mi   R8^Error             ; the input does not exist
move.32   D0, {A6}+analogValue ; analog value

int32     status;
uint32    analogValue, ctrChannel;
...
status = sens_get_ana_value(ctrChannel);
if (status < 0) return -1;
analogValue = (uint32)status;

```



ctr_reserve_channel()

Reserve the battery EEPROM channel.

This system call reserves the channel for a transaction with the EEPROM of the battery. The EEPROM channel is a critical resource which can be shared with other tasks. An error is returned if the channel is busy.

Input (stacking order):

-

Output:

D0	0	Channel reserved and ready to operate.
D0	-1	Channel busy.

Call examples in assembler and C:

```
...  
CALL_BIOS ctr_reserve_channel,0 ; execute the function  
test.32 D0 ;  
jump,mi R8^Error ; wait ...  
  
int32 status;  
...  
status = ctr_reserve_channel();  
if (status < 0) return -1;
```



`ctr_release_channel()`

Release the battery EEPROM channel.

This system call releases the EEPROM channel. The other tasks can now use this channel.

Input (stacking order):

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS ctr_release_channel,0 ; execute the function  
  
...  
ctr_release_channel();
```



ctr_get_eeprom(address)

Read a 16 bit word on the battery EEPROM.

This system call allows to read the battery EEPROM. The address is relative to the beginning of the memory-space.

Input (stacking order):

address Relative address.

Output:

binaryValue 0x0000bbbb.

Call examples in assembler and C:

```
...
push.32   {A6}+address      ;
CALL_BIOS ctr_get_eeprom,1  ; execute the function
move.8    {A6}+binaryValue  ; the value

#define   ioport[10] = 0x10
uint32   binaryValue, *address;
...
address = (uint32 *)ioport;
binaryValue = ctr_get_eeprom(address);
```




ctr_put_eeprom(address, binaryValue)

Write a 16 bit word on the battery EEPROM.

This system call allows to write values on the battery EEPROM. The address is relative to the beginning of the memory-space.

Input (stacking order):

binaryValue 0x0000bbbb.
address Relative address.

Output:

-

Call examples in assembler and C:

```
...  
push.32    {A6}+binaryValue        ; the value  
push.32    {A6}+address            ;  
CALL_BIOS ctr_put_eeprom,2        ; execute the function  
  
uint32    binaryValue, *address;  
...  
ctr_put_eeprom(address, binaryValue);
```



K-Team

<http://www.k-team.com>
info@k-team.com

STR

STR manager (string and ascii conversion manager)

Family ID: 'BIOS'

Table of content

<code>str_reset()</code>	145
<code>str_cvt_dascii_bin32(binary, ascii, paramNb)</code>	146
<code>str_cvt_dascii_bin8(binary, ascii, paramNb)</code>	147
<code>str_cvt_bin32_dascii(ascii, binary, paramNb)</code>	148
<code>str_cvt_bin8_dascii(ascii, binary, paramNb)</code>	149
<code>str_get_size_ascii(ascii)</code>	150
<code>str_skip_parm_protocol(ascii, paramNb)</code>	151
<code>str_cvt_vbin_dascii(ascii, binaryValue)</code>	152
<code>str_skip_parm(ascii, paramNb)</code>	153
<code>str_cvt_vbin_hascii(ascii, binaryValue)</code>	154
<code>str_cvt_hascii_bin32(binary, ascii, paramNb)</code>	155
<code>str_cvt_hascii_bin8(binary, ascii, paramNb)</code>	156
<code>str_cvt_bin32_hascii(ascii, binary, paramNb)</code>	157
<code>str_cvt_bin8_hascii(ascii, binary, paramNb)</code>	158



Generalities

This module operates particular string conversions which can be very useful when we have a connection with visualisation software tools. Formatted ASCII to binary as well as binary to ASCII conversions are realised. Here are general ASCII and binary string formats:

ASCII:

(Sgn) Para. 1, (Sgn) Para. 2, (Sgn) Para. 3,\r\n\0

Ex.

-123, +4567, 234236, 0, -5\r\n\0

Binary:

0xPara. 1 0xPara. 2 0xPara. 3 0xPara. n

Ex.

0x1 0x3 -0x4 0x23



str_reset()

Init of the resources of the manager.

This system call inits the manager.

Input:

-

Output:

-

Call examples in assembler and C:

```
...  
CALL_BIOS str_reset,0      ; execute the function
```

```
...  
str_reset();
```



`str_cnvt_dascii_bin32(binary, ascii, paramNb)`

Conversion from an ASCII (decimal) formatted buffer to a 32-bit binary one.

This system call converts an ASCII (decimal representation) formatted buffer to a 32-bit binary one. An error is returned if the buffer is inconsistent. Here is an example:

ASCII in: 10, 1, 4, 4, 66\r\n\0

Parameter number: 3

Binary out: 0xA, 0x1, 0x4

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>ascii</code>	Pointer on the ASCII buffer.
<code>binary</code>	Pointer on the binary buffer.

Output:

<code>D0</code>	<code>0</code>	Conversion OK.
<code>D0</code>	<code>-1</code>	Format of the buffer inconsistent.

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb      ; number of parameters
push.32  #{A6}+ascii       ; pointer on an ASCII buffer
push.32  #{A6}+binary      ; pointer on a binary buffer
CALL_BIOS str_cnvt_dascii_bin32,3 ; execute the function
test.32  D0                ;
jump,mi  R8^Error         ; buffer format error

```

```

int32    status;
uint32   *binary, paramNb;
char     *ascii;
...
status = str_convert_dascii_bin32(binary, ascii, paramNb);
if (status < 0) return -1;

```



str_cnvt_dascii_bin8(binary, ascii, paramNb)

Conversion from an ASCII (decimal) formatted buffer to an 8-bit binary one.

This system call converts an ASCII (decimal representation) formatted buffer to an 8-bit binary one. An error is returned if the buffer is inconsistent. Here is an example:

ASCII in: 10, 1, 4, 4, 66\r\n\0

Parameter number: 3

Binary out: 0x10, 0x1, 0x4

Input (stacking order):

paramNb	Number of parameters.
ascii	Pointer on the ASCII buffer.
binary	Pointer on the binary buffer.

Output:

D0	0	Conversion OK.
D0	-1	Format of the buffer inconsistent.

Call examples in assembler and C:

```

...
push.32    {A6}+paramNb          ; number of parameters
push.32    #{A6}+ascii           ; pointer on an ASCII buffer
push.32    #{A6}+binary          ; pointer on a binary buffer
CALL_BIOS  str_cnvt_dascii_bin8,3 ; execute the function
test.32    D0                    ;
jump,mi    R8^Error              ; buffer format error

```

```

int32     status;
uint8     *binary;
char      *ascii;
uint32    paramNb;
...
status = str_cnvt_dascii_bin8(binary, ascii, paramNb);
if (status < 0) return -1;

```



`str_cnvt_bin32_dascii(ascii, binary, paramNb)`

Conversion from a 32-bit binary buffer to an ASCII (decimal) formatted one.

This system call converts a 32-bit binary buffer to an ASCII (decimal representation) formatted one. Here is an example:

Binary in: 0xA, 0x1, 0x4, 0x324, 0x345, 0x43

Parameter number: 3

ASCII out: 10, 1, 4\r\n\0

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>binary</code>	Pointer on the binary buffer.
<code>ascii</code>	Pointer on the ASCII buffer.

Output:

-

Call examples in assembler and C:

```

...
push.32   {A6}+paramNb           ; number of parameters
push.32   #{A6}+binary           ; pointer on a binary buffer
push.32   #{A6}+ascii            ; pointer on an ASCII buffer
CALL_BIOS str_cnvt_bin32_dascii,3 ; execute the function

char      *ascii;
uint32    *binary, paramNb;
...
str_convert_bin32_dascii(ascii, binary, paramNb);

```



`str_cnvrt_bin8_dascii(ascii, binary, paramNb)`

Conversion from an 8-bit binary buffer to an ASCII (decimal) formatted one.

This system call converts an 8-bit binary buffer to an ASCII (decimal representation) formatted one. Here is an example:

Binary in: 0xA, 0x1, 0x4, 0x24, 0x35, 0x43

Parameter number: 3

ASCII out: 10, 1, 4\r\n\0

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>binary</code>	Pointer on the binary buffer.
<code>ascii</code>	Pointer on the ASCII buffer.

Output:

-

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb           ; number of parameters
push.32  #{A6}+binary          ; pointer on a binary buffer
push.32  #{A6}+ascii           ; pointer on an ASCII buffer
CALL_BIOS str_cnvrt_bin8_dascii,3 ; execute the function

char     *ascii;
uint8    *binary;
uint32   paramNb;

...
str_cnvrt_bin8_dascii(ascii, binary, paramNb);

```




`str_skip_parm_protocol(ascii, paramNb)`

Skip parameters (and their separators “,”) from an ASCII buffer.

This system call skips parameters from an ASCII buffer. The buffer is formatted as the standard protocol (Para1,Para2, ..). The parameter separator is also skipped. An error is returned if there are too many parameters. Here is an example:

ASCII in: 10, 23455, 4, 456, +23\r\n\0

Parameter number: 4

ASCII out: +23\r\n\0

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>ascii</code>	Pointer on the ASCII buffer.

Output:

<code>ascii</code>	Pointer at the end of the ASCII buffer.
<code>D0</code>	-1 Too many parameters.

Call examples in assembler and C:

```

...
push.32    {A6}+paramNb           ; number of parameters
push.32    #{A6}+ascii            ; pointer on an ASCII buffer
CALL_BIOS  str_skip_parm_protocol,2 ; execute the function
test.32    D0                     ;
jump,mi    R8^Error               ; too many parameters

int32     status;
char      *ascii;
uint32    paramNb;
...
status = str_skip_parm_protocol(ascii, paramNb);
if (status < 0) return -1;

```



`str_cnvt_vbin_dascii(ascii, binaryValue)`

Conversion of a single 32-bit value to an ASCII buffer (decimal mode).

This system call converts a single 32-bit value to a decimal ASCII buffer. Here is an example:

Binary value in: 0x10A34
ASCII out: 68148

Input (stacking order):

`binaryValue` Binary value.
`ascii` Pointer on the ASCII buffer.

Output:

`ascii` Pointer at the end of the ASCII buffer.

Call examples in assembler and C:

```

...
push.32  {A6}+binaryValue      ; binary value
push.32  #{A6}+ascii          ; pointer on an ASCII buffer
CALL_BIOS str_cnvt_vbin_dascii,2 ; execute the function

char     *ascii;
uint32   binaryValue;
...
ascii = str_cnvt_vbin_dascii(ascii, binaryValue);

```



`str_skip_parm(ascii, paramNb)`

Skip parameters (and their separators "SP") from an ASCII buffer.

This system call skips parameters from an ASCII buffer. The buffer is formatted as the standard protocol (Para1 Para2 ..). The parameter separator is also skipped. An error is returned if there are too many parameters. Here is an example:

ASCII in: 10 23455 4 456 +23\r\n\0

Parameter number: 4

ASCII out: +23\r\n\0

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>ascii</code>	Pointer on the ASCII buffer.

Output:

<code>D0</code>	<code>0</code>	Skipping OK.
<code>D0</code>	<code>-1</code>	Too many parameters.

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb      ; number of parameters
push.32  #{A6}+ascii      ; pointer on an ASCII buffer
CALL_BIOS str_skip_parm,2  ; execute the function
test.32  D0                ;
jump,mi  R8^Error         ; too many parameters

int32    status;
char     *ascii;
uint32   paramNb;
...
status = str_skip_parm(ascii, paramNb);
if (status < 0) return -1;

```



`str_cnvt_vbin_hascii(ascii, binaryValue)`

Conversion of a single 32-bit value to an ASCII buffer (hexadecimal mode).

This system call converts a single 32-bit value to an hexadecimal ASCII buffer. Here is an example:

Binary value in: 0x10A34
ASCII out: 10A34

Input (stacking order):

`binaryValue` Binary value.
`ascii` Pointer on the ASCII buffer.

Output:

`ascii` Pointer at the end of the ASCII buffer.

Call examples in assembler and C:

```
...
push.32  {A6}+binaryValue      ; binary value
push.32  #{A6}+ascii          ; pointer on an ASCII buffer
CALL_BIOS str_cnvt_vbin_hascii,2 ; execute the function

char     *ascii;
uint32   binaryValue;
...
ascii = str_cnvt_vbin_hascii(ascii, binaryValue);
```



`str_cnvt_hascii_bin32(binary, ascii, paramNb)`

Conversion from an ASCII (hexadecimal) formatted buffer to a 32-bit binary one.

This system call converts an ASCII (hexadecimal representation) formatted buffer to a 32-bit binary one. An error is returned if the buffer is inconsistent. Here is an example:

ASCII in: 10, 1, 4, 4, 66\r\n\0

Parameter number: 3

Binary out: 0x10, 0x1, 0x4

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>ascii</code>	Pointer on the ASCII buffer.
<code>binary</code>	Pointer on the binary buffer.

Output:

<code>D0</code>	<code>0</code>	Conversion OK.
<code>D0</code>	<code>-1</code>	Format of the buffer inconsistent.

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb      ; number of parameters
push.32  #{A6}+ascii      ; pointer on an ASCII buffer
push.32  #{A6}+binary     ; pointer on a binary buffer
CALL_BIOS str_cnvt_hascii_bin32,3 ; execute the function
test.32  D0                ;
jump,mi  R8^Error        ; buffer format error

int32    status;
uint32   *binary, paramNb;
char     *ascii;
...
status = str_convert_hascii_bin32(binary, ascii, paramNb);
if (status < 0) return -1;

```



`str_cnvrt_hascii_bin8(binary, ascii, paramNb)`

Conversion from an ASCII (hexadecimal) formatted buffer to an 8-bit binary one.

This system call converts an ASCII (hexadecimal representation) formatted buffer to an 8-bit binary one. An error is returned if the buffer is inconsistent. Here is an example:

ASCII in: 10, 1, 4, 4, 66\r\n\0

Parameter number: 3

Binary out: 0x10, 0x1, 0x4

Input (stacking order):

paramNb		Number of parameters.
ascii		Pointer on the ASCII buffer.
binary		Pointer on the binary buffer.

Output:

D0	0	Conversion OK.
D0	-1	Format of the buffer inconsistent.

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb          ; number of parameters
push.32  #{A6}+ascii           ; pointer on an ASCII buffer
push.32  #{A6}+binary          ; pointer on a binary buffer
CALL_BIOS str_cnvrt_hascii_bin8,3 ; execute the function
test.32  D0                    ;
jump,mi  R8^Error              ; buffer format error

```

```

int32    status;
uint8    *binary;
char     *ascii;
uint32   paramNb;
...
status = str_cnvrt_hascii_bin8(binary, ascii, paramNb);
if (status < 0) return -1;

```



`str_cnvt_bin32_hascii(ascii, binary, paramNb)`

Conversion from a 32-bit binary buffer to an ASCII (hexadecimal) formatted one.

This system call converts a 32-bit binary buffer to an ASCII (hexadecimal representation) formatted one. Here is an example:

Binary in: 0xA, 0x1, 0x4, 0x324, 0x345, 0x43

Parameter number: 3

ASCII out: A, 1, 4\r\n\0

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>binary</code>	Pointer on the binary buffer.
<code>ascii</code>	Pointer on the ASCII buffer.

Output:

-

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb          ; number of parameters
push.32  #{A6}+binary          ; pointer on a binary buffer
push.32  #{A6}+ascii           ; pointer on an ASCII buffer
CALL_BIOS str_cnvt_bin32_hascii,3 ; execute the function

char     *ascii;
uint32   *binary, paramNb;
...
str_convert_bin32_hascii(ascii, binary, paramNb);

```




`str_cnvrt_bin8_hascii(ascii, binary, paramNb)`

Conversion from an 8-bit binary buffer to an ASCII (hexadecimal) formatted one.

This system call converts an 8-bit binary buffer to an ASCII (hexadecimal representation) formatted one. Here is an example:

Binary in: 0xA, 0x1, 0x4, 0x24, 0x35, 0x43

Parameter number: 3

ASCII out: A, 1, 4\r\n\0

Input (stacking order):

<code>paramNb</code>	Number of parameters.
<code>binary</code>	Pointer on the binary buffer.
<code>ascii</code>	Pointer on the ASCII buffer.

Output:

-

Call examples in assembler and C:

```

...
push.32  {A6}+paramNb          ; number of parameters
push.32  #{A6}+binary          ; pointer on a binary buffer
push.32  #{A6}+ascii           ; pointer on an ASCII buffer
CALL_BIOS str_cnvrt_bin8_hascii,3 ; execute the function

char     *ascii;
uint8    *binary;
uint32   paramNb;
...
str_cnvrt_bin8_hascii(ascii, binary, paramNb);

```



References

- [JDN86] "Common Assembly Language for Microprocessors", Jean-Daniel Nicoud and Patrick Fäh, Swiss Federal Institute of Technology (LAMI), December 1986.
- [Mot89] "Central Processor Unit Reference Manual", Ref. CPU32RM/AD, Motorola INC., 1989.
- [Mot91] "MC68331 User's Manual", Ref. MC68331UM/AD, Motorola INC., 1991.
- [Mot92] "Programmer's Reference Manual", Ref. M68000PM/AD Rev. 1, Motorola INC., 1992.
- [Fra91] "Système multiprocesseur pour la commande de robots", Report R91.50, Edoardo Franzi, Swiss Federal Institute of Technology (LAMI), August 1991.