BIOS MANUAL

Version 5.01

K-Team

Lausanne, 19 March 1999

Documentation author

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

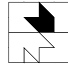
email: info@k-team.com
WWW: http://www.k-team.com

Trademark Acknowledgements

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

NOTICE:

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

K-system 

All information in this document
is preliminary and subject to change

Table of content

Abstract 2
Preliminary 2
General constraints 4
Rules for building applications 4

BIOS 7
Table of content 7
Generalities 8

COM 14
Table of content 14
Generalities 15

TIM 23
Table of content 23
Generalities 24

SENS 43
Table of content 43
Generalities 44

MSG 47
Table of content 47
Generalities 48

VAR 55
Table of content 55
Generalities 56

SER 69
Table of content 69
Generalities 70

STR 79
Table of content 79
Generalities 80

MMA 95
Table of content 95
Generalities 96

References 116
C examples 117

Franzi Edo.
K-Team S.A.
franzi@k-team.com

K376SBC BIOS 1.0 Reference Manual

Rev. 1.00

Abstract

The high level of complexity of the K376SBC board 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 MC68376 microcontroller hardware features [Mot89][Mot91][Mot92]. The program examples are shown in C and in assembler CALM[1] syntax [JDN86].

At the end of this document some examples will be shown to make the work with K376SBC 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 SER controls the serial SCI channel resources, module MSG controls the network communications, TIM controls the multitasking, etc.).

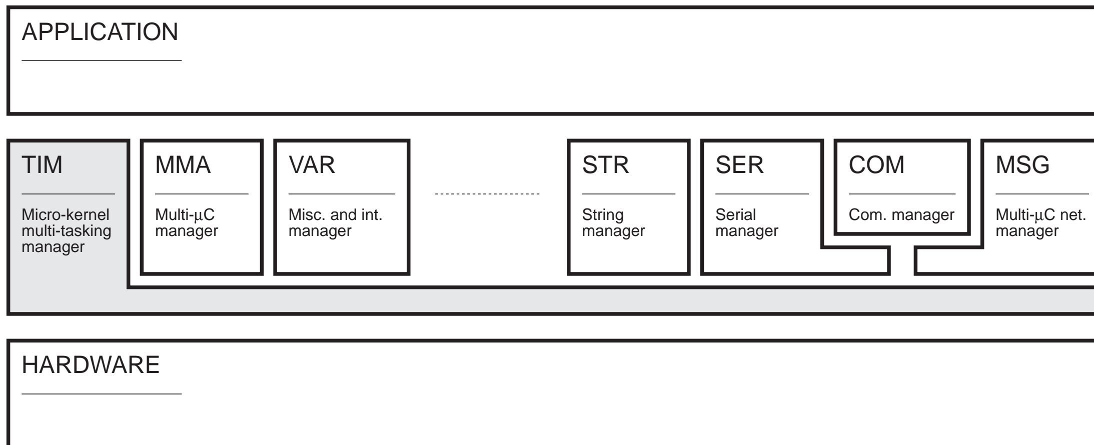


Figure 1: General topology of the BIOS

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.
- COM: I/Os communication manager.
- MMA: multi-microcontroller manager.
- SENS: A/D converter manager.
- MSG: multi-microcontroller communication manager.
- VAR: misc device manager (jumpers, LEDs, etc.).
- TIM: multitasking manager.
- SER: serial channel SCI manager.
- USART: serial channel USART manager.
- STR: string manager.

General constraints

It is vital to observe the few rules below in order to realise robust applications. None of the K376SBC 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 K376SBC 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

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

2. Launching and killing processes

The BIOS does not include any memory management. For this reason and 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 ressources

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

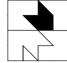
```
/* Writing process */
...
tim_lock();     /* This locks the time sharing   */
for (i = 0; i < KNBELEMENTS; i++)
     vector[i] = i;

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 ressources

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(“K376SBC board\n”);
RELEASE_COM     /* This releases the standard I/Os channel */
```

K-system 

All information in this document
is preliminary and subject to change

BIOS Rev. 1.00

Franzi Edo.
K-Team S.A.
franzi@k-team.com

BIOS

BIOS manager

Family ID: 'BIOS'

Table of content

bios_reset() 9
bios_get_ident() 10
bios_get_rev() 11
bios_get_system() 12
bios_restart_system() .13
bios_get_node() 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 manager. DO NOT USE FOR APPLICATIONS!

This system call inits all the common resources used by the different BIOS managers. 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 the ASCII string (terminated with ‘\0’) identifier 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.

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();
```

bios_get_node()

Get the node identifier in network operations.

This system call returns the node identifier of the board. Each identifier is unique.

Input (stacking order):

-

Output:

D0	node	Node identifier.

Call examples in assembler and C:

```
...
CALL_BIOS  bios_get_node,0      ; execute the function
move.32    D0,(A6)+node         ; node identifier

uint32     node;
...
node = bios_get_node();
```

COM Rev. 1.02

Franzi Edo.
K-Team S.A.
franzi@k-team.com

COM

COM manager (i/o manager)

Family ID: 'BIOS'

Table of content

com_reset() 18
com_reserve_channel() 19
com_release_channel() 20
com_send_buffer(buffer, size) 21
com_receive_byte() 22
com_get_status_channel() 23

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.

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 that 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()

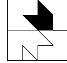
Get the status of the active channel.

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

```
Input (stacking order):
-
Output:
D0        +xxx00000001    Tx buffer not empty
D0        +xxx00000010    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();
```

K-system 

All information in this document
is preliminary and subject to change

TIM Rev. 1.00

Franzi Edo.
K-Team S.A.
franzi@k-team.com

TIM

TIM manager (multi-tasking kernel)

Family ID: ‘BIOS’

Table of content

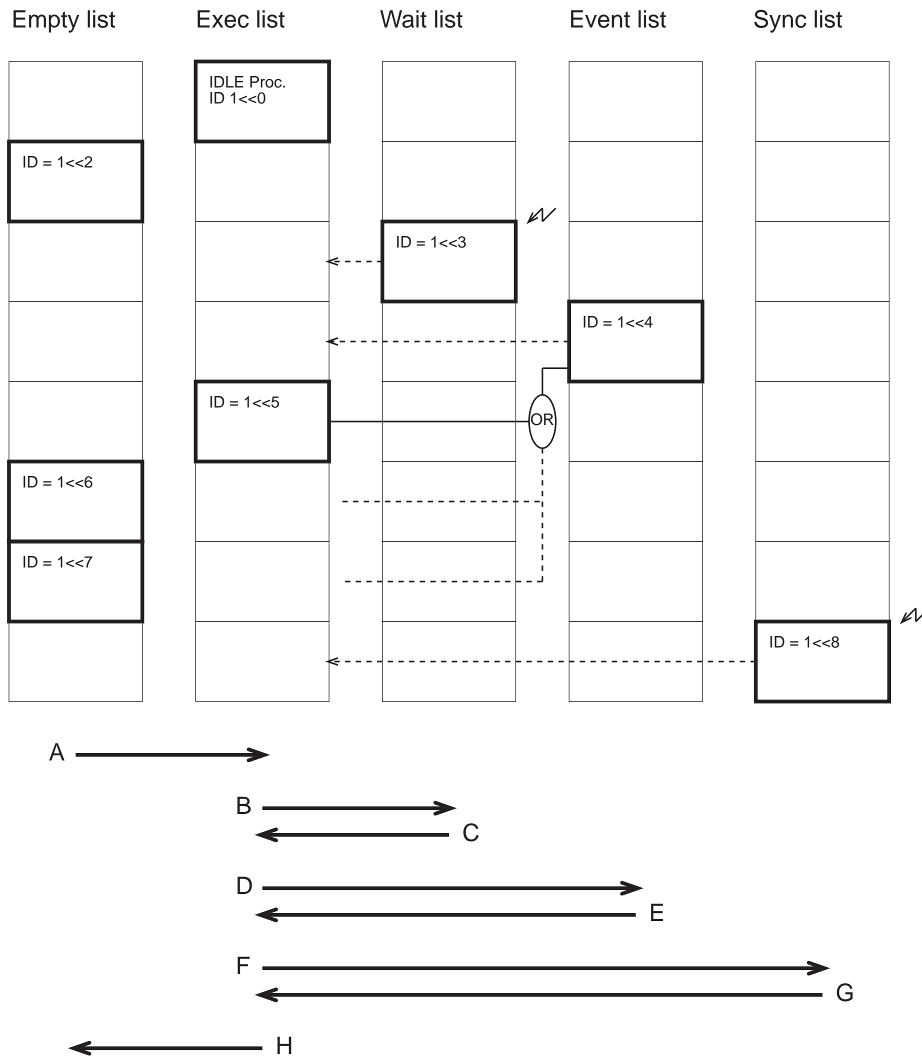
tim_reset() 27
tim_new_inst_task(textId, stack, procedure) 28
tim_remove_inst_task(id) 29
tim_suspend_task(time) .30
tim_generate_event() 31
tim_wait_event(taskMask) 32
tim_get_id() 33
tim_get_ticcount() 34
tim_run_kernel() 35
tim_switch_fast() 36
tim_lock() 37
tim_unlock() 38
tim_define_association(reference, general) 39
tim_find_association(reference) 40
tim_remove_association(reference) 41
tim_wait_sync(syncMask) .42
tim_get_task_des_ptr() .43

Generalities

This module manages the multitasking capabilities of the system. Thirty-two user tasks can be run simultaneously. The first task descriptor[1] 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 another 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 listt

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 minimun 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):

procedure		Pointer on the task code.
stack		Pointer on the stack.
textId		Pointer on a string terminated with null '\0'.

Output:

D0	2**0..2**31	ID of the task descriptor.
D0	-1	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_suspend_task(time)

Suspend a task for a time.

This system call suspends the current task for a time. The time can be chosen inside an interval of 1 ms to about 50 days (32-bits) with 1 ms of resolution!

Input (stacking order):

time — Length of time.

Output:

-

Call examples in assembler and C:

```
...
push.32    {A6}+time            ; length of time
CALL_BIOS  tim_suspend_task,1   ; execute the function

uint32     time;
...
tim_suspend_task(time);
```

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):

syncMask= 2**2	Message sent by MSG manager.
syncMask= 2**3	Message received by MSG manager.
syncMask= 2**4	Message sent by SER manager.
syncMask= 2**5	Byte received by SER manager.
syncMask= 2**8	IRQ interruption.
syncMask= 2**9	Message sent by MMA manager.
syncMask= 2**10	Message received by MMA manager.
syncMask= 2**11	Message sent by USART manager.
syncMask= 2**12	Byte received by USART manager.

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):

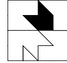
-

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-system 

All information in this document
is preliminary and subject to change

SENS Rev. 1.01

Franzi Edo.
K-Team S.A.
franzi@k-team.com

SENS

SENS manager (IR sensors and analog manager)

Family ID: 'BIOS'

Table of content

sens_reset() 66
sens_get_reflected_value(sensorNb) 67
sens_get_ambient_value(sensorNb) 68
sens_get_pointer() 69
sens_get_ana_value(inputNb) 70

Generalities

This module manages the sixteen analog inputs. The hardware which controls the different phases of the IR sensor measure is completely under the control of this module.

- Channels 0 to 14: user.
- Channel 15: is used to measure the voltage of the input power supply.

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_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. Sixteen channels are used:

0..14:	User	4 mV/bit..
15:	Voltage of the input power supply	20 mV/bit.

Input (stacking order):

inputNb		Number of the analog input [0..15].

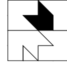
Output:

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;
```

K-system 

All information in this document
is preliminary and subject to change

Franzi Edo.
K-Team S.A.
franzi@k-team.com

MSG

MSG manager (local multi-microcontroller network manager)

Family ID: ‘BIOS’

Table of content

msg_reset() 72
msg_reserve_channel(channelNb) 73
msg_release_channel(channelNb) 74
msg_send_message(mesgS, sizeS) 75
msg_receive_message(mesgR, sizeR) 76
msg_snd_rec_message(msgS, sizeS, msgR, sizeR, rep) 77

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 3 shows how a message is coded and the different phases of the protocol.

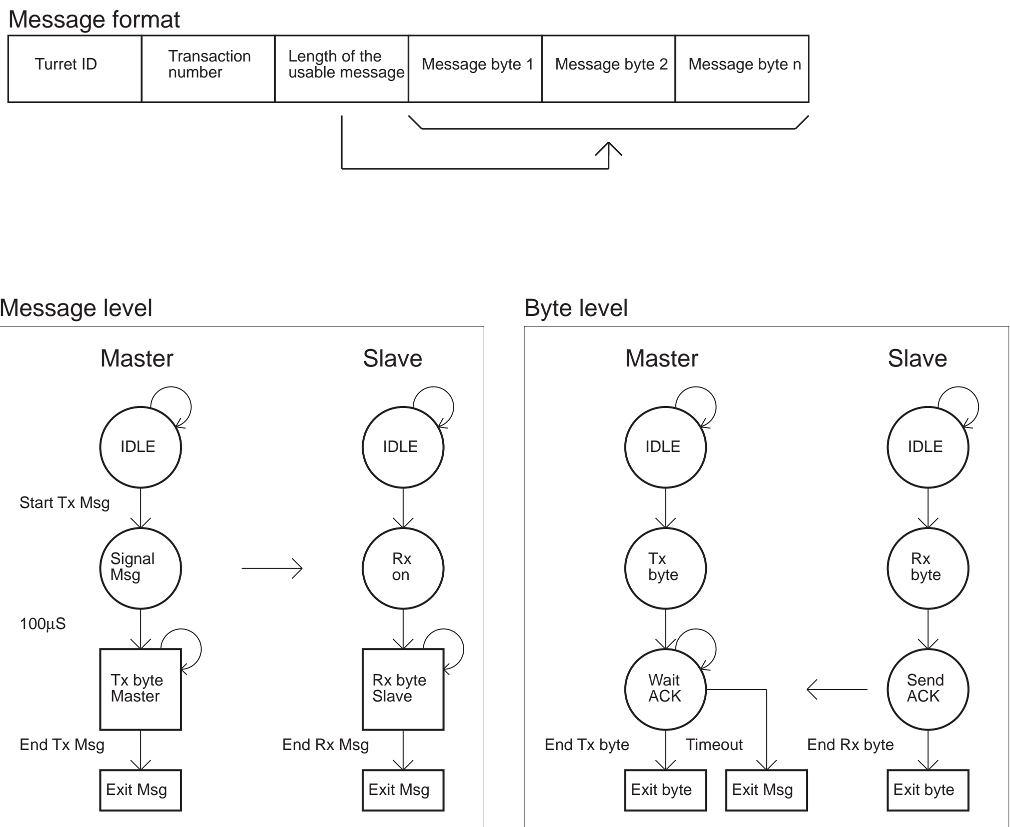


Figure 3: Message format and protocol sequences

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    {A6}+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 (stautus < 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 (stautus < 0) return -1;
```

msg_send_message(mesgS, 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):		
sizeS		Size of the message.
messageS		Pointer on the message.
Output:		
D0	0	Message sent correctly.
D0	-1	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(mesgR, 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):

sizeR		Size of the message.
messageR		Pointer on the message.
Output:		
D0	0	Message received correctly.
D0	-1	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):

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

Output:

D0	0	Message sent and received correctly.
D0	-1	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 (stautus < 0) return -1;
```

Franzi Edo.
K-Team S.A.
franzi@k-team.com

VAR

VAR manager (misc. and interruption manager)

Family ID: 'BIOS'

Table of content

var_reset() 81
var_get_jumper() 82
var_on_led(ledNb) 83
var_off_led(ledNb) 84
var_change_led(ledNb) 85
var_set_irq_vector(procedure) 86
var_enable_irq() 87
var_disable_irq() 88
var_set_exception(procedure, exceptionNb) 89
var_cpu_speed(CPUSpeedValue) 90
var_get_extension(address) 91
var_put_extension(address, binaryValue) 92

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.

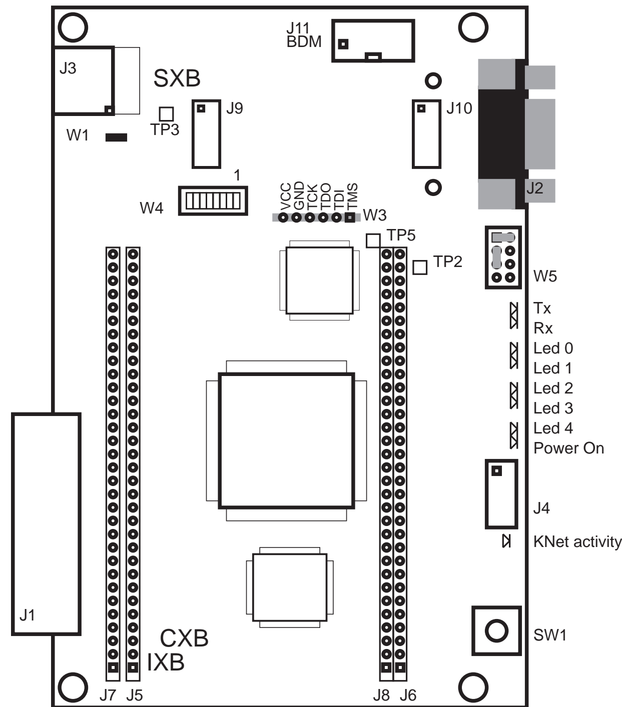


Figure 4: LED and jumper definitions

var_reset()

Init of the resources of the module.

This system call inits the hardware and turns off all the LEDs.

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 jumpers.

This system call returns the state values of the three jumpers 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     {A6}+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     {A6}+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_change_led(ledNb)

Change the state of one LED.

This system call toggles the state of one 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     {A6}+ledNb          ;
CALL_BIOS   var_change_led,1    ; execute the function
test.32     D0                  ;
jump,mi     R8^Error            ; the LED does not exist

int32       status;
uint32      ledNb;
...
status = var_change_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(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):

exeptionNb		Number of the exception.
procedure		Pointer on the procedure.
Output:		
D0	0	OK
D0	-1	The exception does not exist

Call examples in assembler and C:

```
...
push.32     {A6}+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_get_extension(address)

Read on the extension bus.

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

Input (stacking order):

address	Relative address [0..63].

Output:

binaryValue	0x000000bb.

Call examples in assembler and C:

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

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

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):

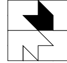
binaryValue	0x000000bb.
address	Relative address [0..63].

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);
```

K-system 

All information in this document
is preliminary and subject to change

SER Rev. 1.00

Franzi Edo.
K-Team S.A.
franzi@k-team.com

SER

SER manager (serial RS232 manager)

Family ID: 'BIOS'

Table of content

ser_reset() 94
ser_reserve_channel() 95
ser_release_channel() 96
ser_config(baudrate) 97
ser_send_buffer(buffer, size) 98
ser_receive_byte() 99
ser_tx_status() 100
ser_rx_status() 101

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     {A6}+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 that 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):

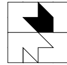
-

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;
```

K-system 

All information in this document
is preliminary and subject to change

STR Rev. 1.00

Franzi Edo.
K-Team S.A.
franzi@k-team.com

STR

STR manager (string and ascii conversion manager)

Family ID: 'BIOS'

Table of content

str_reset() 104
str_cnvt_dascii_bin32(binary, ascii, paramNb) 105
str_cnvt_dascii_bin8(binary, ascii, paramNb) 106
str_cnvt_bin32_dascii(ascii, binary, paramNb) 107
str_cnvt_bin8_dascii(ascii, binary, paramNb) 108
str_get_size_ascii(ascii) 109
str_skip_parm_protocol(ascii, paramNb) 110
str_cnvt_vbin_dascii(ascii, binaryValue) 111
str_skip_parm(ascii, paramNb) 112
str_cnvt_vbin_hascii(ascii, binaryValue) 113
str_cnvt_hascii_bin32(binary, ascii, paramNb) 114
str_cnvt_hascii_bin8(binary, ascii, paramNb) 115
str_cnvt_bin32_hascii(ascii, binary, paramNb) 116
str_cnvt_bin8_hascii(ascii, binary, paramNb) 117

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):

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_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):

paramNb	Number of parameters.
binary	Pointer on the binary buffer.
ascii	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_cnvt_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):

paramNb	Number of parameters.
binary	Pointer on the binary buffer.
ascii	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_bin8_dascii,3     ; execute the function

char        *ascii;
uint8       *binary;
uint32      paramNb;
...
str_cnvt_bin8_dascii(ascii, binary, paramNb);
```

str_get_size_ascii(ascii)

Get the size of an ASCII buffer.

This system call returns the size of an ASCII buffer. The character ‘\0’ is not counted.

Input (stacking order):	
ascii	Pointer on the ASCII buffer.
Output:	
size	Size of the buffer.

Call examples in assembler and C:

```
...
push.32    #{A6}+ascii          ; pointer on an ASCII buffer
CALL_BIOS  str_get_size_ascii,1 ; execute the function
move.32    D0,{A6}+size         ; the value

uint32     size;
char       *ascii;
...
size = str_get_size_ascii(ascii);
```

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):

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

Output:

ascii		Pointer at the end of the ASCII buffer.
D0	-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):

paramNb		Number of parameters.
ascii		Pointer on the ASCII buffer.
Output:		
D0	0	Skipping OK.
D0	-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,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):

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_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_cnvt_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_cnvt_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_cnvt_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):

paramNb	Number of parameters.
binary	Pointer on the binary buffer.
ascii	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_cnvt_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):

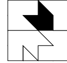
paramNb	Number of parameters.
binary	Pointer on the binary buffer.
ascii	Pointer on the ASCII buffer.

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_bin8_hascii,3    ; execute the function

char        *ascii;
uint8       *binary;
uint32      paramNb;
...
str_cnvt_bin8_hascii(ascii, binary, paramNb);
```

K-system 

All information in this document
is preliminary and subject to change

MMA Rev. 1.00

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() 97
mma_reserve_channel_0() .98
mma_reserve_channel_1() .99
mma_reserve_channel_2() .100
mma_release_channel_0() .101
mma_release_channel_1() .102
mma_release_channel_2() .103
mma_send_buffer_0(buffer, size) 104
mma_send_buffer_1(buffer, size) 105
mma_send_buffer_2(buffer, size) 106
mma_receive_byte_0() 107
mma_receive_byte_1() 108
mma_receive_byte_2() 109
mma_tx_status_0() 110
mma_tx_status_1() 111
mma_tx_status_2() 112
mma_rx_status_0() 113
mma_rx_status_1() 114
mma_rx_status_2() 115

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 that 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 that 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 that 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;
```

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.

C examples

Here are some reference documents and examples of programs written in CALM assembler and in C.

Assembler files

- The REF_BIOS.ASI.
- The REF_COM.ASI.
- The MACRO.ASI.
- Example 0: A simple CALM assembler application showing how to launch some processes.

C files

- The k376SBCsys.h.
- Example 0: A simple C application showing how to launch some processes.