Foenix Toolbox Programmer's Guide

Firmware functions for the Foenix Retro System F256 computers

Peter Weingartner

September 28, 2024

Contents

Chapter 1

Introduction

Copyright Information

Foenix Toolbox and all code except for the FatFS file system library are published under the BSD 3 Clause License. Please see the source code for the license terms. The Foenix Toolbox file system is provided by the FatFS file system, which is covered under its own license. For information about the author of FatFS and its license terms, please see the Foenix Toolbox source code.

Chapter 2

Devices

Devices on the Foenix computers fall into one of two main categories: channel devices, and block devices.

Channel Devices

Channel devices are predominantly sequential, byte oriented devices. They are essentially byte streams. A program can read or write a series of bytes from or to the device. A channel can have the notion of a "cursor" which represents the point where a read or write will happen. Examples of channel devices include the console, the serial ports, and files.

Currently, the only fully supported channel devices are open files, the keyboard, and the screen. In the future, there should be full support for the serial ports, the parallel port, and the MIDI ports. Channel devices are assigned as follows:

Number	Device
0	Main console (keyboard and main screen or channel B)
1	Secondary console (keyboard and EVID or channel A)
2	Serial Port 1
3	Serial Port 2
5	MIDI Ports
6	Files

By default, channels 0 and 1 are open automatically to devices 0 and 1 respectively at boot time.

Block Devices

Block devices organize their data into blocks of bytes. A block may be read from or written to a block device, and blocks maybe accessed in any order desired. The F256K2e comes with two block devices: the internal and external SD cards.

Out of the box, there are three block devices supported by Foenix Toolbox:

Number	Device
0	sd0—External SD card
1	sd1—Internal SD card

File Channels

Files represent a special channel pseudo-device. Although files are stored on block devices, they may be open as file channels, which may be accessed like a channel device. There is a special file channel driver, which converts channel reads and writes on a file to the appropriate block calls. Access to these file channels is managed in part through the file system calls listed below.

Paths

File and directory names follow the Unix style path conventions. That is, the forward slash (/) is used as a separator, and drives are treated as directories ("/sd", "/hd", *etc.*). FAT32 long file names are supported, but not Unicode characters. Special path names "." and "..". are supported to specify a path relative to the current path. Example paths are:

```
/sd0/hello.txt
/sd1/system/format.elf
../games/HauntedCastle/start
```

Chapter 3

Toolbox Functions

3.1 Channel Functions

Example: C

Example: Assembler

sys_chan_read_b – 0xFFE024

Read a single character from the channel. Returns the character, or 0 if none are available.

Prototype	**short** sys_chan_read_b(**short** channel)
channel	the number of the channel
Returns	the value read (if negative, error)

Example: C

```
// Read a byte from channel #0 (keyboard)
short b = sys_chan_read_b(0);
if (b >= 0) {
    // We have valid data from 0–255 in b
}
```

Example: Assembler

```
lda #0                  ; Select channel #0
jsl sys_chan_read_b     ; Read from the channel
bit #$ffff              ; If negative...
bmi error               ; Process an error

; We have valid data in A
```

sys_chan_read – 0xFFE028

Read bytes from a channel and fill a buffer with them, given the number of the channel and the size of the buffer. Returns the number of bytes read.

Prototype	**short** sys_chan_read(**short** channel, **unsigned char** ∗ buffer, **short** size)
channel	the number of the channel
buffer	the buffer into which to copy the channel data
size	the size of the buffer.
Returns	number of bytes read, any negative number is an error code

Example: C

```
char buffer[80];
short n = sys_chan_read(0, buffer, 80);
if (n >= 0) {
    // We correctly read n bytes into the buffer
    } else {
    // We have an error
}
```

Example: Assembler

```
pei #80             ; Push the size of the buffer

pei #‘buffer        ; Push the address of the buffer
pei #<>buffer

lda #0              ; Select channel #0

jsl sys_chan_read   ; Try to read the bytes from the channel

ply                 ; Clean up the stack
ply
ply

bit #$ffff          ; If result is negative...
bmi error           ; Go to process the error

sta n               ; Otherwise: save the number of bytes read
```

sys_chan_readline – 0xFFE02C

Read a line of text from a channel (terminated by a newline character or by the end of the buffer). Returns the number of bytes read.

Prototype	**short** sys_chan_readline(**short** channel, **unsigned char** ∗ buffer, **short** size)
channel	the number of the channel
buffer	the buffer into which to copy the channel data
size	the size of the buffer
Returns	number of bytes read, any negative number is an error code

Example: C

```
short c = ...; // The channel number
unsigned char buffer[128];
short n = sys_chan_read_line(c, buffer, 128);
```

Example: Assembler

```
pei #128               ; Push the size of the buffer
pei #‘buffer           ; Push the pointer to the buffer
pei #<>buffer
lda c                  ; Set the channel number to read from
jsl sys_chan_read_line ; Attempt to read a line from the console

ply                    ; Clean up the stack
ply
ply

sta n                  ; Save the number of bytes read
```

sys_chan_write_b – 0xFFE030

Write a single byte to the channel.

Prototype	**short** sys_chan_write_b(**short** channel, uint8_t b)
channel	the number of the channel
b	the byte to write
Returns	0 on success, a negative value on error

Example: C

```
// Write ‘a’ to the console
short result = sys_chan_write_b(0, ’a’);
```

Example: Assembler

```
    pei #'a'                ; Push 'a' as the b parameter
    lda #0                  ; Select the console (channel #0)
    jsl sys_chan_write_b    ; Write the character to the console
    ply                     ; Clean up the stack
```

sys_chan_write – 0xFFE034

Write bytes from a buffer to a channel, given the number of the channel and the size of the buffer. Returns the number of bytes written.

Prototype	**short** sys_chan_write(**short** channel, **const** uint8_t ∗ buffer, **short** size)
channel	the number of the channel
buffer	
size	
Returns	number of bytes written, any negative number is an error code

Example: C

```
char ∗ message = ’Hello, world!\n’;
short n = sys_chan_write(0, message, strlen(message));
```

Example: Assembler

```
    pei #15               ; Push the size of the buffer
    pei #`message         ; Push the pointer to the message
    pei #<>message
    lda #0                ; Select the console (channel #0)
    jsl sys_chan_write    ; Write the buffer to the console

    ply                   ; Clean up the stack
    ply
    ply

    ; ...
message:
    .null "Hello, world!", 13, 10
```

sys_chan_status – 0xFFE038

Gets the status of the channel. The meaning of the status bits is channel-specific, but four bits are recommended as standard:

- 0x01: The channel has reached the end of its data
- 0x02: The channel has encountered an error
- 0x04: The channel has data that can be read
- 0x08: The channel can accept data

Prototype	**short** sys_chan_status(**short** channel)
channel	the number of the channel
Returns	the status of the device

Example: C

```
// Check the status of the file_in channel
short status = sys_chan_status(file_in);
if (status & 0x01) {
    // We have reached end of file
}
```

Example: Assembler

```
lda file_in
jsl sys_chan_status
and #$01
beq have_data
; We have reached end of file
```

sys_chan_flush – 0xFFE03C

Ensure any pending writes to a channel are completed.

Prototype	**short** sys_chan_flush(**short** channel)
channel	the number of the channel
Returns	0 on success, any negative number is an error code

Example: C

```
short file_out = ...; // Channel number
sys_chan_flush(file_out); // Flush the channel
```

Example: Assembler

```
lda file_out          ; Channel number
jsl sys_chan_flush    ; Flush the channel
```

sys_chan_seek – 0xFFE040

Set the position of the input/output cursor. This function may not be honored by a given channel as not all channels are "seekable." In addition to the usual channel parameter, the function takes two other parameters:

- position: the new position for the cursor
- base: whether the position is absolute (0), or relative to the current position (1).

Prototype	**short** sys_chan_seek(**short** channel, **long** position, **short** base)
channel	the number of the channel
position	the position of the cursor
base	whether the position is absolute or relative to the current position
Returns	0 = success, a negative number is an error.

Example: C

short c = ...; *// The channel number*
sys_chan_seek(c, −10, 1); *// Move the point back 10 bytes*

Example: Assembler

```
pei #1              ; Move is relative
pei #-10            ; Move back by 10 bytes
lda c               ; Select the channel
jsl sys_chan_seek   ; Move the channel cursor

ply                 ; Clean up the stack
ply
```

sys_chan_ioctrl – 0xFFE044

Send a command to a channel. The mapping of commands and their actions are channel-specific. The return value is also channel and command-specific. The `buffer` and `size` parameters provide additional data to the commands, what exactly needs to go in them (if anything) is command-specific. Some commands require data in the buffer, and others do not.

Prototype	**short** sys_chan_ioctrl(**short** channel, **short** command, uint8_t ∗ buffer, **short** size)
channel	the number of the channel
command	the number of the command to send
buffer	pointer to bytes of additional data for the command
size	the size of the buffer
Returns	0 on success, any negative number is an error code

Example: C

short c = ...; *// The channel number*
short cmd = ...; *// The command*
short r = sys_chan_ioctrl(c, cmd, 0, 0); *// Send simple command*

Example: Assembler

```
    pei #0                  ; Push 0 for the size
    pei #0                  ; Push the null pointer for the buffer
    pei #0
    lda cmd                 ; Push the command
    pha
    lda c
    jsl sys_chan_ioctrl     ; Issue the command

    ply                     ; Clean up the stack
    ply
    ply
    ply
```

sys_chan_open – 0xFFE048

Open a channel device for reading or writing given: the number of the device, the path to the resource on the device (if any), and the access mode. The access mode is a bitfield:

- 0x01: Open for reading
- 0x02: Open for writing
- 0x03: Open for reading and writing

Prototype	**short** sys_chan_open(**short** dev, **const char** ∗ path, **short** mode)
dev	the device number to have a channel opened
path	a ”path” describing how the device is to be open
mode	s the device to be read, written, both
Returns	the number of the channel opened, negative number on error

Example: C

```
// Serial port: 9600bps, 8−data bits, 1 stop bit, no parity
short chan = sys_chan_open(2, "9600,8,1,N", 3);
```

Example: Assembler

```
    pei #3                ; Mode: Read & Write
    pei #`path            ; Pointer to the path
    pei #<>path
    lda #2                ; Device #2 (UART)
    jsl sys_chan_open     ; Open the channel to the UART

    ply                   ; Clean up the stack
    ply
    ply

    ; ...

path:
    .null "9600,8,1,N"
```

sys_chan_close – 0xFFE04C

Close a channel that was previously open by sys_chan_open.

Prototype	**short** sys_chan_close(**short** chan)
chan	the number of the channel to close
Returns	nothing useful

Example: C

```
short c = ...; // The channel number
sys_chan_close(c); // Close the channel
```

Example: Assembler

```
lda c               ; Get the channel number
jsl sys_chan_close  ; Close the channel
```

sys_chan_swap – 0xFFE050

Swaps two channels, given their IDs. If before the call, channel ID `channel1` refers to the file "hello.txt", and channel ID `channel2` is the console, then after the call, `channel1` is the console, and `channel2` is the open file "hello.txt". Any context for the channels is preserved (for instance, the position of the file cursor in an open file).

Prototype	**short** sys_chan_swap(**short** channel1, **short** channel2)
channel1	the ID of one of the channels
channel2	the ID of the other channel
Returns	0 on success, any other number is an error

sys_chan_device – 0xFFE054

Given a channel ID (the only parameter), return the ID of the device associated with the channel. The channel must be open.

Prototype	**short** sys_chan_device(**short** channel)
channel	the ID of the channel to query
Returns	the ID of the device associated with the channel, negative number for error

3.2 Block Device Functions

sys_bdev_register – 0xFFE05C

Prototype	**short** sys_bdev_register(p_dev_block device)
device	pointer to the description of the device to register
Returns	0 on succes, negative number on error

sys_bdev_read – 0xFFE060

Prototype	**short** sys_bdev_read(**short** dev, **long** lba, uint8_t * buffer, **short** size)
dev	the number of the device
lba	the logical block address of the block to read
buffer	the buffer into which to copy the block data
size	the size of the buffer.
Returns	number of bytes read, any negative number is an error code

sys_bdev_write – 0xFFE064

Prototype	**short** sys_bdev_write(**short** dev, **long** lba, **const** uint8_t * buffer, **short** size)
dev	the number of the device
lba	the logical block address of the block to write
buffer	the buffer containing the data to write
size	the size of the buffer.
Returns	number of bytes written, any negative number is an error code

sys_bdev_status – 0xFFE068

Prototype	**short** sys_bdev_status(**short** dev)
dev	the number of the device
Returns	the status of the device

sys_bdev_flush – 0xFFE06C

Prototype	**short** sys_bdev_flush(**short** dev)
dev	the number of the device
Returns	0 on success, any negative number is an error code

sys_bdev_ioctrl – 0xFFE070

Prototype	**short** sys_bdev_ioctrl(**short** dev, **short** command, uint8_t * buffer, **short** size)
dev	the number of the device
command	the number of the command to send
buffer	pointer to bytes of additional data for the command
size	the size of the buffer
Returns	0 on success, any negative number is an error code

3.3 File System Functions

sys_fsys_open – 0xFFE074

Prototype	**short** sys_fsys_open(**const char** ∗ path, **short** mode)
path	the ASCIIZ string containing the path to the file.
mode	the mode (e.g. r/w/create)
Returns	the channel ID for the open file (negative if error)

sys_fsys_close – 0xFFE078

Prototype	**short** sys_fsys_close(**short** fd)
fd	the channel ID for the file
Returns	0 on success, negative number on failure

sys_fsys_opendir – 0xFFE07C

Prototype	**short** sys_fsys_opendir(**const char** ∗ path)
path	the path to the directory to open
Returns	the handle to the directory if ¿= 0. An error if ¡ 0

sys_fsys_closedir – 0xFFE080

Prototype	**short** sys_fsys_closedir(**short** dir)
dir	the directory handle to close
Returns	0 on success, negative number on error

sys_fsys_readdir – 0xFFE084

Prototype	**short** sys_fsys_readdir(**short** dir, p_file_info file)
dir	the handle of the open directory
file	pointer to the t_file_info structure to fill out.
Returns	0 on success, negative number on failure

sys_fsys_findfirst – 0xFFE088

Prototype	**short** sys_fsys_findfirst(**const char** ∗ path, **const char** ∗ pattern, p_file_info file)
path	the path to the directory to search
pattern	the file name pattern to search for
file	pointer to the t_file_info structure to fill out
Returns	the directory handle to use for subsequent calls if ¿= 0, error if negative

sys_fsys_findnext – 0xFFE08C

Prototype	**short** sys_fsys_findnext(**short** dir, p_file_info file)
dir	the handle to the directory (returned by fsys_findfirst) to search
file	pointer to the t_file_info structure to fill out
Returns	0 on success, error if negative

sys_fsys_get_label – 0xFFE090

Prototype	**short** sys_fsys_get_label(**const char** * path, **char** * label)
path	path to the drive
label	buffer that will hold the label... should be at least 35 bytes
Returns	0 on success, error if negative

sys_fsys_set_label – 0xFFE094

Prototype	**short** sys_fsys_set_label(**short** drive, **const char** * label)
drive	drive number
label	buffer that holds the label
Returns	0 on success, error if negative

sys_fsys_mkdir – 0xFFE098

Prototype	**short** sys_fsys_mkdir(**const char** * path)
path	the path of the directory to create.
Returns	0 on success, negative number on failure.

sys_fsys_delete – 0xFFE09C

Prototype	**short** sys_fsys_delete(**const char** * path)
path	the path of the file or directory to delete.
Returns	0 on success, negative number on failure.

sys_fsys_rename – 0xFFE0A0

Prototype	**short** sys_fsys_rename(**const char** * old_path, **const char** * new_path)
old_path	he current path to the file
new_path	the new path for the file
Returns	0 on success, negative number on failure.

sys_fsys_set_cwd – 0xFFE0A4

Prototype	**short** sys_fsys_set_cwd(**const char** * path)
path	the path that should be the new current
Returns	0 on success, negative number on failure.

sys_fsys_get_cwd – 0xFFE0A8

Prototype	**short** sys_fsys_get_cwd(**char** * path, **short** size)
path	the buffer in which to store the directory
size	the size of the buffer in bytes
Returns	0 on success, negative number on failure.

sys_fsys_load – 0xFFE0AC

Prototype	**short** sys_fsys_load(**const char** * path, uint32_t destination, uint32_t * start)
path	the path to the file to load
destination	the destination address (0 for use file's address)
start	pointer to the long variable to fill with the starting address
Returns	0 on success, negative number on error

sys_fsys_register_loader – 0xFFE0B0

Prototype	**short** sys_fsys_register_loader(**const char** ∗ extension, p_file_loader loader)
extension	the file extension to map to
loader	pointer to the file load routine to add
Returns	0 on success, negative number on error

sys_fsys_stat – 0xFFE0B4

Prototype	**short** sys_fsys_stat(**const char** ∗ path, p_file_info file)
path	the path to the file to check
file	pointer to a file info record to fill in, if the file is found.
Returns	0 on success, negative number on error

3.4 Text System Functions

sys_txt_set_mode – 0xFFE0E0

Prototype	**short** sys_txt_set_mode(**short** screen, **short** mode)
screen	the number of the text device
mode	a bitfield of desired display mode options
Returns	0 on success, any other number means the mode is invalid for the screen

sys_txt_set_xy – 0xFFE0E8

Prototype	**void** sys_txt_set_xy(**short** screen, **short** x, **short** y)
screen	the number of the text device
x	the column for the cursor
y	the row for the cursor

sys_txt_get_xy – 0xFFE0EC

Prototype	**void** sys_txt_get_xy(**short** screen, p_point position)
screen	the number of the text device
position	pointer to a t_point record to fill out

sys_txt_get_region – 0xFFE0F0

Prototype	**short** sys_txt_get_region(**short** screen, p_rect region)
screen	the number of the text device
region	pointer to a t_rect describing the rectangular region (using character cells for size and size)
Returns	0 on success, any other number means the region was invalid

sys_txt_set_region – 0xFFE0F4

Prototype	**short** sys_txt_set_region(**short** screen, p_rect region)
screen	the number of the text device
region	pointer to a t_rect describing the rectangular region (using character cells for size and size)
Returns	0 on success, any other number means the region was invalid

sys_txt_set_color – 0xFFE0F8

Prototype	**void** sys_txt_set_color(**short** screen, **unsigned char** foreground, **unsigned char** background)
screen	the number of the text device
foreground	the Text LUT index of the new current foreground color (0 - 15)
background	the Text LUT index of the new current background color (0 - 15)

sys_txt_get_color – 0xFFE0FC

Prototype	**void** sys_txt_get_color(**short** screen, **unsigned char** ∗ foreground, **unsigned char** ∗ background)
screen	the number of the text device
foreground	the Text LUT index of the new current foreground color (0 - 15)
background	the Text LUT index of the new current background color (0 - 15)

sys_txt_set_cursor_visible – 0xFFE100

Prototype	**void** sys_txt_set_cursor_visible(**short** screen, **short** is_visible)
screen	the screen number 0 for channel A, 1 for channel B
is_visible	TRUE if the cursor should be visible, FALSE (0) otherwise

sys_txt_set_font – 0xFFE104

Prototype	**short** sys_txt_set_font(**short** screen, **short** width, **short** height, **unsigned char** * data)
screen	the number of the text device
width	width of a character in pixels
height	of a character in pixels
data	pointer to the raw font data to be loaded

sys_txt_setsizes – 0xFFE0E4

Prototype	**void** sys_txt_setsizes(**short** chan)
chan	

sys_txt_get_sizes – 0xFFE108

Prototype	**void** sys_txt_get_sizes(**short** screen, p_extent text_size, p_extent pixel_size)
screen	the screen number 0 for channel A, 1 for channel B
text_size	the size of the screen in visible characters (may be null)
pixel_size	the size of the screen in pixels (may be null)

sys_txt_set_border – 0xFFE10C

Prototype	**void** sys_txt_set_border(**short** screen, **short** width, **short** height)
screen	the number of the text device
width	the horizontal size of one side of the border (0 - 32 pixels)
height	the vertical size of one side of the border (0 - 32 pixels)

sys_txt_set_border_color – 0xFFE110

Prototype	**void** sys_txt_set_border_color(**short** screen, **unsigned char** red, **unsigned char** green, **unsigned char** blue)
screen	the number of the text device
red	the red component of the color (0 - 255)
green	the green component of the color (0 - 255)
blue	the blue component of the color (0 - 255)

sys_txt_put – 0xFFE114

Prototype	**void** sys_txt_put(**short** screen, **char** c)
screen	the number of the text device
c	the character to print

sys_txt_print – 0xFFE118

Prototype	**void** sys_txt_print(**short** screen, **const char** * message)
screen	the number of the text device
message	the ASCII Z string to print

3.5 Interrupt Functions

sys_int_enable_all – 0xFFE004

This function enables all maskable interrupts at the CPU level. It returns a system-dependent code that represents the previous level of interrupt masking. Note: this does not change the mask status of interrupts in the machine's interrupt controller, it just changes if the CPU ignores IRQs or not.

Prototype	**void** sys_int_enable_all()

Example: C

```
// Enable processing of IRQs
sys_int_enable_all();
```

Example: Assembler

```
; Enable processing of IRQs
jrl sys_int_enable_all
```

sys_int_disable_all – 0xFFE008

This function disables all maskable interrupts at the CPU level. It returns a system-dependent code that represents the previous level of interrupt masking. Note: this does not change the mask status of interrupts in the machine's interrupt controller, it just changes if the CPU ignores IRQs or not.

Prototype	**void** sys_int_disable_all()

Example: C

```
// Disable processing of IRQs
sys_int_disable_all();
```

Example: Assembler

```
; Disable processing of IRQs
jrl sys_int_disable_all
```

sys_int_disable – 0xFFE00C

This function disables a particular interrupt at the level of the interrupt controller. The argument passed is the number of the interrupt to disable.

Prototype	**void** sys_int_disable(**unsigned short** n)
n	the number of the interrupt: n[7..4] = group number, n[3..0] = individual number.

Example: C

```
// Disable the start–of–frame interrupt
sys_int_disable(INT_SOF_A);
```

Example: Assembler

```
    lda #INT_SOF_A        ; Enable the start-of-frame interrupt
    jsl sys_int_disable
```

sys_int_enable – 0xFFE010

This function enables a particular interrupt at the level of the interrupt controller. The argument passed is the number of the interrupt to enable. Note that interrupts that are enabled at this level will still be disabled, if interrupts are disabled globally by `sys_int_disable_all`.

Prototype	**void** sys_int_enable(**unsigned short** n)
n	the number of the interrupt

Example: C

```
// Enable the start-of-frame interrupt
sys_int_enable(INT_SOF_A);
```

Example: Assembler

```
    lda #INT_SOF_A        ; Enable the start-of-frame interrupt
    jsl sys_int_enable
```

sys_int_register – 0xFFE014

Registers a function as an interrupt handler. An interrupt handler is a function which takes and returns no arguments and will be run in at an elevated privilege level during the interrupt handling cycle.

The first argument is the number of the interrupt to handle, the second argument is a pointer to the interrupt handler to register. Registering a null pointer as an interrupt handler will "deregister" the old handler.

The function returns the handler that was previously registered.

Prototype	p_int_handler sys_int_register(**unsigned short** n, p_int_handler handler)
n	the number of the interrupt
handler	pointer to the interrupt handler to register
Returns	the pointer to the previous interrupt handler

Example: C

```
// Handler for the start-of-frame interrupt
// Must be a far sub-routine (returns through RTL)
__attribute__((far)) void sof_handler() {
        // Interrupt handler code here...
}

// Register a handler for the start-of-frame interrupt
p_int_handler old = sys_int_register(INT_SOF_A, sof_handler);
```

Example: Assember

```
; Handler for the start-of-frame interrupt
; Must be a far sub-routine (returns through RTL)
sof_handler:
    ; Handler code here...
    rtl

    ; Code to register the handler...
    pei #‘sof_handler       ; push pointer to sof_handler
    pei #<>sof_handler

    lda #INT_SOF_A          ; A = the number for the SOF_A interrupt

    jsl sys_int_register

    ply                     ; Clean up the stack
    ply

    sta old                 ; Save the pointer to the old handler
    stx old+2
```

sys_int_pending – 0xFFE018

Query an interrupt to see if it is pending in the interrupt controller. NOTE: User programs will probably never need to use this call, since it is handled by the Toolbox itself.

Prototype	**short** sys_int_pending(**unsigned short** n)
n	the number of the interrupt: n[7..4] = group number, n[3..0] = individual number.
Returns	non-zero if interrupt n is pending, 0 if not

Example: C

```
// Check to see if start−of−frame interrupt is pending
short is_pending = sys_int_pending(INT_SOF_A);
if (is_pending) {
        // The interrupt has not yet been acknowledged
}
```

Example: Assembler

```
    ; Check to see if the start-of-frame interrupt is pending
    lda #INT_SOF_A
    jsl sys_int_pending
    cmp #0
    beq sof_not_pending

    ; Code for when start-of-frame is pending

sof_not_pending:
```

sys_int_clear – 0xFFE020

This function acknowledges the processing of an interrupt by clearing its pending flag in the interrupt controller. NOTE: User programs will probably never need to use this call, since it is handled by the Toolbox itself.

Prototype	**void** sys_int_clear(**unsigned short** n)
n	the number of the interrupt: n[7..4] = group number, n[3..0] = individual number.

Example: C

```
// Acknowledge the processing of the start−of−frame interrupt
sys_int_clear(INT_SOF_A);
```

Example: Assembler

```
; Acknowledge the processing of the start-of-frame interrupt
lda #INT_SOF_A
jsl sys_int_clear
```

3.6 General Functions

sys_proc_exit – 0xFFE000

This function ends the currently running program and returns control to the command line. It takes a single short argument, which is the result code that should be passed back to the kernel. This function does not return.

void sys_proc_exit(**short** result)

result	the code to return to the kernel

Example: C

```
sys_proc_exit(0); // Quit the program with a result code of 0
```

Example: Assembler

```
lda #0               ; Return code of 0
jsl sys_proc_exit    ; Quit the program
```

sys_proc_run – 0xFFE0D8

Prototype	**short** sys_proc_run(**const char** * path, **int** argc, **char** * argv[])
path	the path to the executable file
argc	the number of arguments passed
argv	the array of string arguments
Returns	the return result of the program

sys_get_info – 0xFFE01C

Prototype	**void** sys_get_info(p_sys_info info)
info	pointer to a s_sys_info structure to fill out

sys_mem_get_ramtop – 0xFFE0B8

Prototype	uint32_t sys_mem_get_ramtop()
Returns	the address of the first byte of reserved system RAM (one above the last byte the user program can use)

sys_mem_reserve – 0xFFE0BC

Prototype	uint32_t sys_mem_reserve(uint32_t bytes)
bytes	the number of bytes to reserve
Returns	address of the first byte of the reserved block

sys_time_jiffies – 0xFFE0C0

Prototype	uint32_t sys_time_jiffies()
Returns	the number of jiffies since the last reset

sys_rtc_set_time – 0xFFE0C4

Prototype	**void** sys_rtc_set_time(p_time time)
time	pointer to a t_time record containing the correct time

sys_rtc_get_time – 0xFFE0C8

Prototype	**void** sys_rtc_get_time(p_time time)
time	pointer to a t_time record in which to put the current time

sys_kbd_scancode – 0xFFE0CC

Prototype	uint16_t sys_kbd_scancode()
Returns	the next scan code from the keyboard... 0 if nothing pending

sys_kbd_layout – 0xFFE0D4

Prototype	**short** sys_kbd_layout(**const char** ∗ tables)
tables	pointer to the keyboard translation tables
Returns	0 on success, negative number on error