CircleOS V4.6

Conception document

Document version
25 November 2013

Contents

1. INTRODUCTION....7

1.1 Purpose of this document....7
1.2 Scope of this document....7
1.3 Additional help or information....7

2. SOFTWARE ARCHITECTURE....8

2.1 Program....8
2.2 Memory (STM32)....10
2.2.1 Flash....10
2.2.2 RAM....16
2.3 Memory (STM8)....20
2.3.1 ROM....20
2.3.2 RAM....21
2.4 Project files organization....21
2.5 Software configuration....24
2.5.1 Shared declarations....24
2.5.2 Specific declarations....24

3. LCD....26

3.1 Files....26
3.2 Principles....26
3.2.1 Fonts....26
3.2.2 Character fonts....27
3.2.3 Images....27
3.3 Functions....30
3.3.1 Internals Functions....30
3.3.2 APIs....30
3.4 Structures....31

4. AUDIO....32

4.1 File....32
4.2 Principles....32
4.2.1 Generalities....32

4.2.2 I2S implementation....32
4.2.3 I2C implementation....32
4.3 Functions....33
4.3.1 Internal functions....33
4.3.2 APIs....35
4.4 Structures....36

5. TOUCHSCREEN....37

5.1 Files....37
5.2 Principle....37
5.3 Functions....37
5.3.1 Internal functions....37
5.3.2 APIs....38
5.4 Structures....38
5.5 Calibration principle....39
5.5.1 Conversion....39
5.5.2 Correction....39
5.5.3 Calibration....39
5.5.4 Median filter....40

6. TOOLBAR....41

6.1 Files....41
6.2 Functions....41
6.2.1 Internal functions....41
6.2.2 APIs....42
6.3 Structures....42

7. BUTTON / JOYSTICK....43

7.1 Files....43
7.2 Functions....43
7.2.1 Internal functions....43
7.2.2 APIs....43
7.3 Structures....44

8. POWER....45

8.1 File....45
8.2 Power management principle....45

8.2.1 Power states....45
8.2.2 Standby management....46
8.3 Functions....46
8.3.1 Internal functions....46
8.3.2 APIs....46
8.4 Structures....46

9. LIST....47

9.1 Files....47
9.2 Principles....47
9.2.1 List....47
9.3 Functions....48
9.3.1 Internal functions....48
9.3.2 APIs....49
9.4 Structures....49

10. MENU....50

10.1 Files....50
10.2 Principles....50
10.3 Functions....52
10.3.1 Internal functions....52
10.3.2 APIs....54
10.4 Structures....55

11. FILE SYSTEM....57

11.1 Introduction....57
11.2 CircleOS < 4.4....57
11.2.1 DOSFS presentation....57
11.2.2 Structures....58
11.2.3 APIs....60
11.3 CircleOS >= 4.4....61
11.3.1 FatFs presentation....61
11.3.2 Configuration....62
11.3.3 Stuctures....63
11.3.4 New APIs....63
11.4 Example....63

12. UTILITIES....64

12.1 Files....64
12.2 Various APIs....64
12.3 Save screen....66
12.4 High priority timer interrupt handling....66
12.4.1 Principle....66
12.4.2 Example 1: Timeout....66
12.4.3 Example 2: Periodic call....67

13. CX EXTENSION....68

13.1 Introduction....68
13.2 Platform pin out....69
13.2.1 Primer2....69
13.2.2 Open4 STM32E / STM32G....70
13.2.3 Open4 STM32C....71
13.2.4 Open4 STM32L....72
13.2.5 Open4 STM3240G....73
13.2.6 Open4 STM32429x....74
13.3 Files....75
13.4 Principles....75
13.5 Using GPIO's....76
13.5.1 Configuration....76
13.5.2 Reading a pin....76
13.5.3 Writing a pin....76
13.6 Using the ADC....78
13.6.1 Configuration....78
13.6.2 Reading a value....78
13.7 Using the USART....79
13.7.1 Configuration....79
13.7.2 Receiving characters....79
13.7.3 Transmitting characters....80
13.8 Using the SPI....82
13.8.1 Configuration....82
13.8.2 Receiving characters....83
13.8.3 Transmitting characters....84
13.8.4 Example....85
13.9 Using the I²C....86
13.10 Examples....86

14. DMA2D (CHROM-ART ACCELERATOR).....87
14.1 Presentation.....87
14.2 DMA2D Integration in STM32F429 EvoPrimer.....87
14.2.1 The DMA2D hardware.....87
14.2.2 Large Internal Memory.....88
14.2.3 SDRAM Controller.....88
14.2.4 Image of the Screen in RAM.....88
14.3 Handling Bitmaps.....88
14.3.1 Display of a picture.....88
14.3.2 Overlaying Bitmaps.....90
14.3.3 Moving Objects.....90
14.4 Transparency.....91
14.4.1 Alpha component for the pixel.....91
14.4.2 Global alpha constant.....92
14.4.3 Blending.....92
14.5 Editable objects.....93
14.5.1 Dynamic vs Read Only Objects.....93
14.5.2 Editing a dynamic object.....93
14.6 Advanced features.....94
14.6.1 Z-Order.....94
14.6.2 Touchscreen and objects.....94
14.7 Files.....94
14.8 Functions.....95
14.8.1 Internal functions.....95
14.8.2 API's.....95
14.8.3 Structures and variables.....98
14.9 Examples.....101

15. BACKUP REGISTERS.....102
15.1 Registers saved at power off.....102
15.2 SYS2 register details.....103

16. GLOSSARY.....104

1. Introduction

The STM32 Primer2 is an innovative, low-cost evaluation and development package that is designed to provide a fun and easy introduction to the features of the STM32 with an ARM Cortex™-M3 core.

It was developed by Raisonance for ST Microelectronics, in 2007.

Based on the previous STM32 Primer (released in 2007), Raisonance developed the new STM32 Primer2, with more user interface options, longer lasting Li-Ion power supply, more hardware peripherals and an STM32F103V with more memory than before (512 KB of Flash ROM, 64 KB of RAM).

The Open4 is the last Primer consisting on a Open base and a target board, which can be equipped with a STM32 or a STM8L micro controller.

The CircleOS is a simple operating system, that mainly provides handlers for the hardware interface, scheduling capacity, and basic APIs, that can be called by user applications, developed by ST's customers.

1.1 Purpose of this document

This document recalls some generalities and then details precisely the implementation of the functionalities added to the CircleOS software.

1.2 Scope of this document

This document is applicable to all the Primer versions.

1.3 Additional help or information

Please visit the Raisonance website: *http://www.raisonance.com/* and the forum *http://www.raisonance.com/Forum/punbb/* or contact Raisonance.

Address: Raisonance S.A.S.
17, Avenue Jean Kuntzmann,
38330 Montbonnot Saint Martin
France

Email: ***support@raisonance.com***

If you find any errors or omissions, or if you have suggestions for improving this manual, please let us know by email.

2. Software architecture

2.1 Program

In the CircleOS, the main program (main.c) and its *main()* function, are only designed to initialize the devices and the interrupts. Then handlers have to be called regularly. Each handler is a function designed to manage a Primer hardware interface or functionality.

The handlers are scheduled by the `SysTickHandler`, in the order defined in the Handler table.

According to `SysTick_Configuration()`, the `SysTickHandler` is called around every 1ms (IT timer, depends of the PLL pace). So each handler is called also around every 1ms.

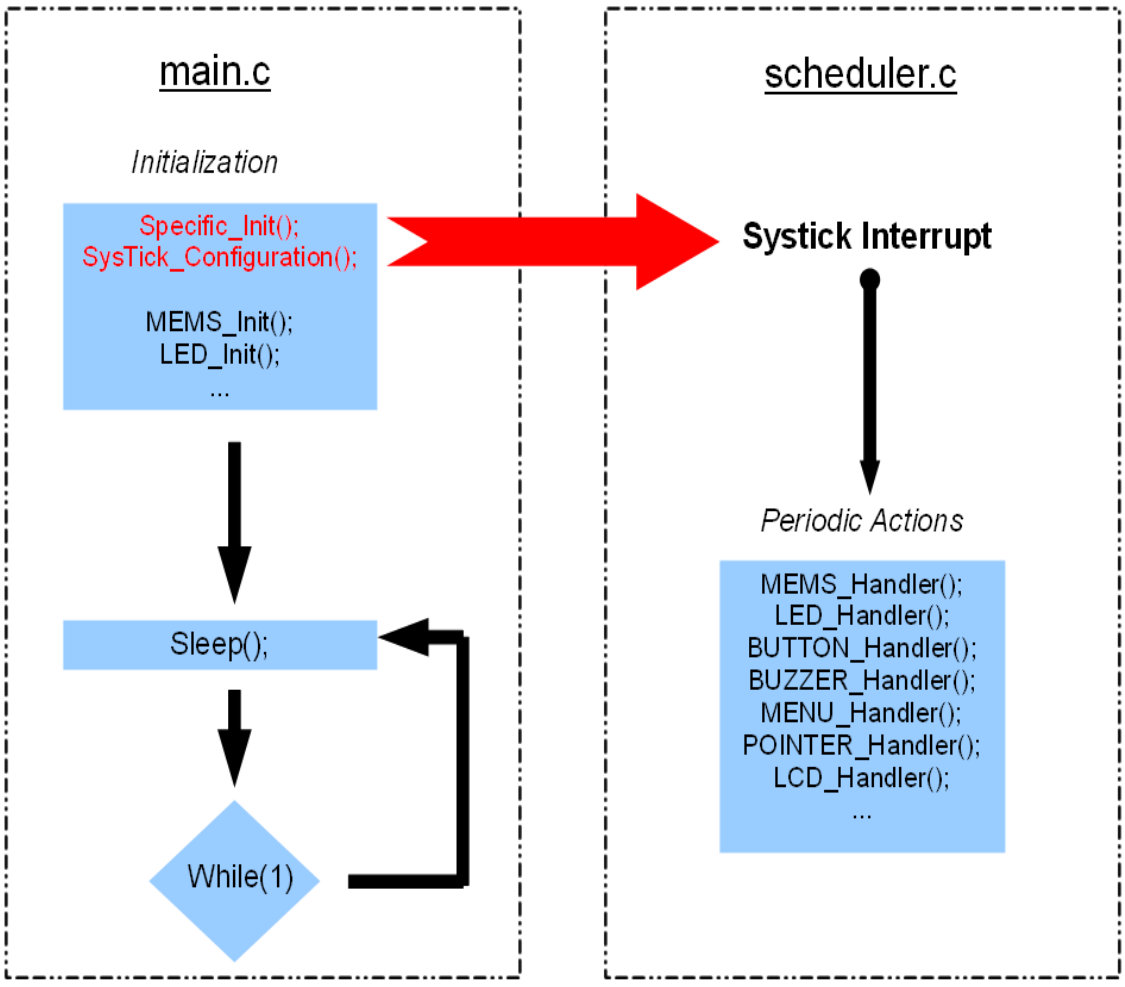
Table of handlers (there can be up to 16 handlers):

- MEMS_Handler,
- LED_Handler,
- BUTTON_Handler,
- BUZZER_Handler,
- MENU_Handler,
- POINTER_Handler,
- LCD_Handler,
- DRAW_Handler,
- RTC_DisplayTime,
- AUDIO_Handler,
- TOUCHSCR_Handler,
- TOOLBAR_Handler,
- POWER_Handler.

Applications developed by users (final customers) can be linked with the CirceOs, loaded into the Primer, and can use the APIs provided by the CircleOS to virtualize hardware accesses.

The MEMS acquisition function is called by the TIM2 interrupt, and has a higher priority than the other handlers. User applications are launched by the `MENU_Handler`.

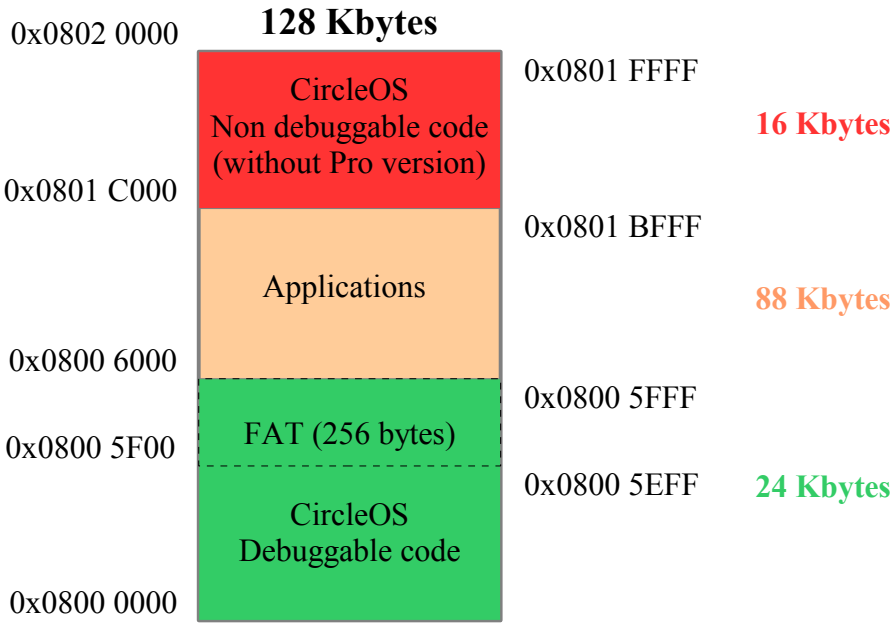
The list of APIs provided for user applications is located in “circle_api.h”.


main.c
Initialization
Specific_Init();
SysTick_Configuration();
MEMS_Init();
LED_Init();
...
Sleep();
While(1)
scheduler.c
Systick Interrupt
Periodic Actions
MEMS_Handler();
LED_Handler();
BUTTON_Handler();
BUZZER_Handler();
MENU_Handler();
POINTER_Handler();
LCD_Handler();
...

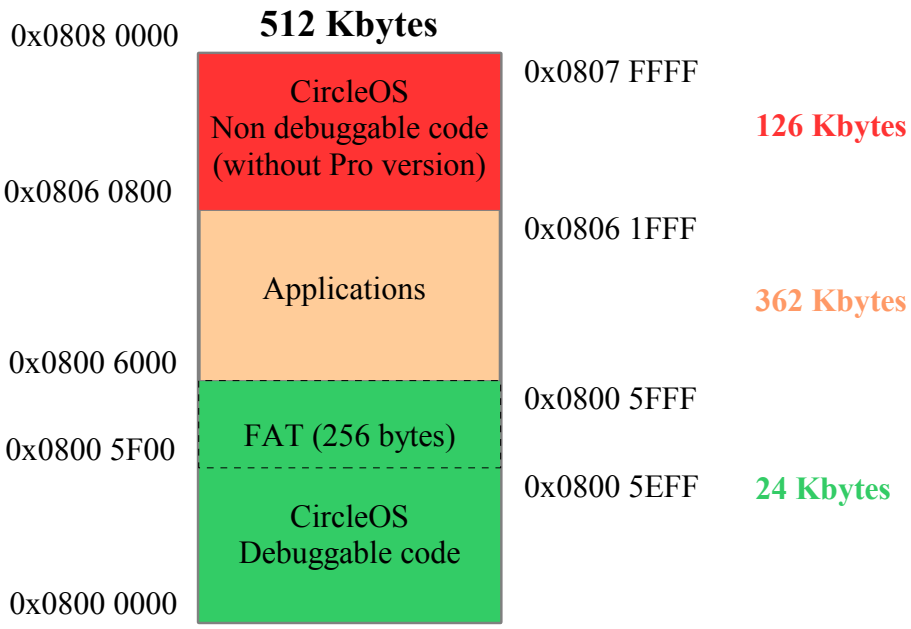
2.2 Memory (STM32)

2.2.1 Flash

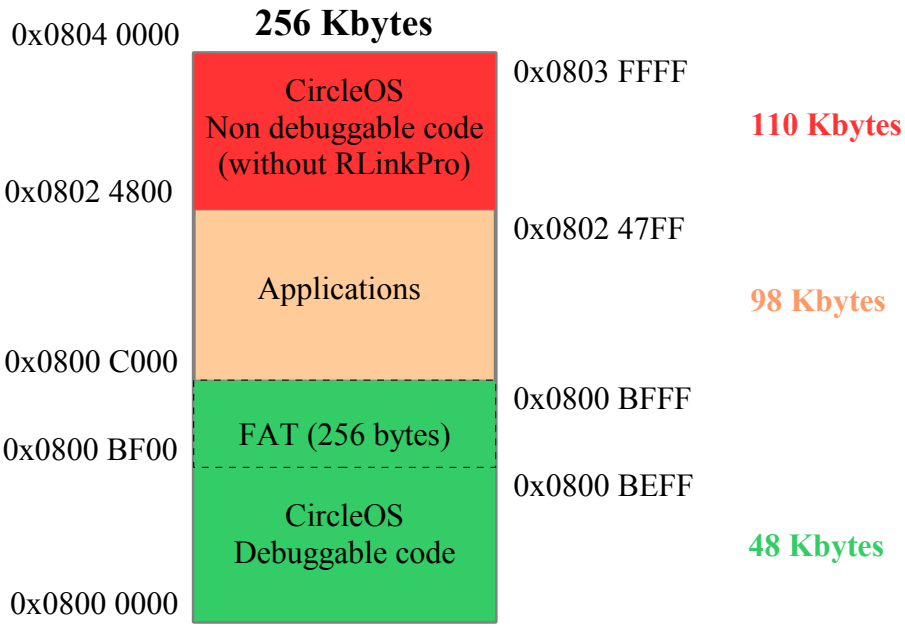
Primer 1:



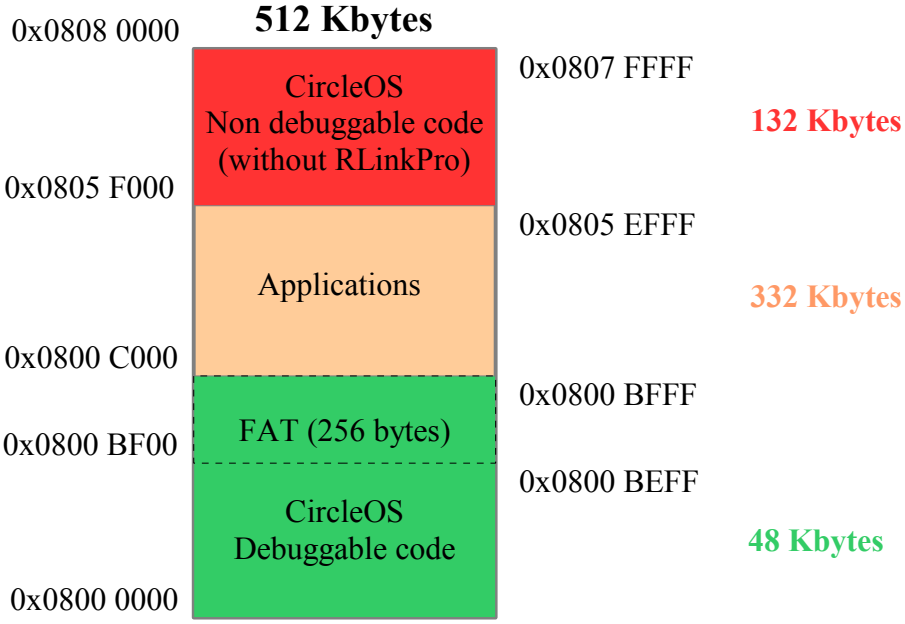
Primer 2:



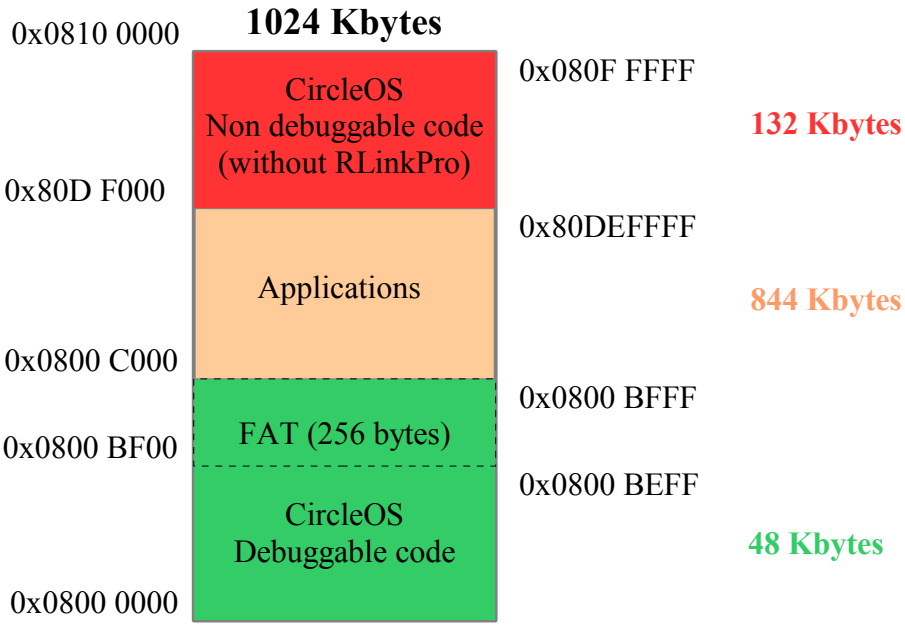
Open4 (STM32C):



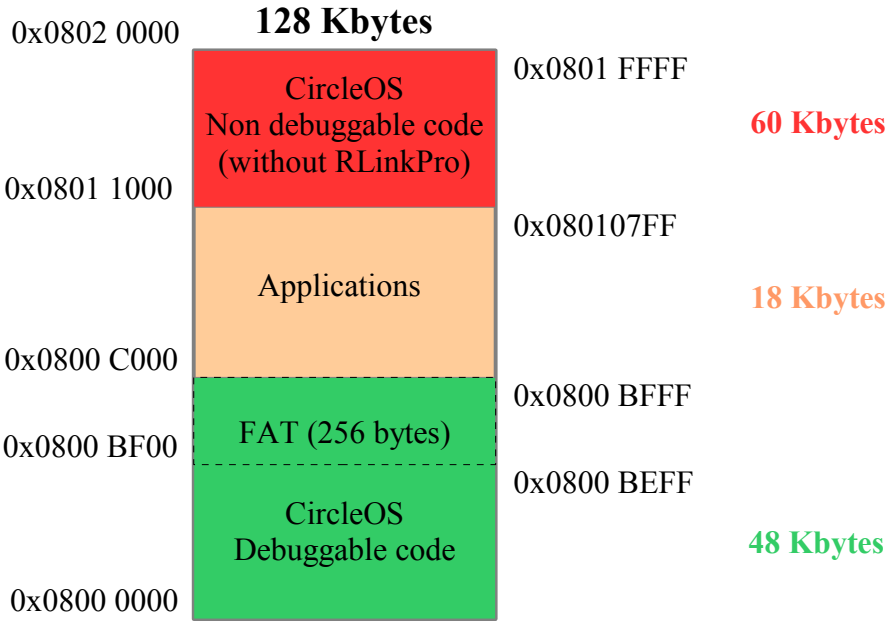
Open4 (STM32E):



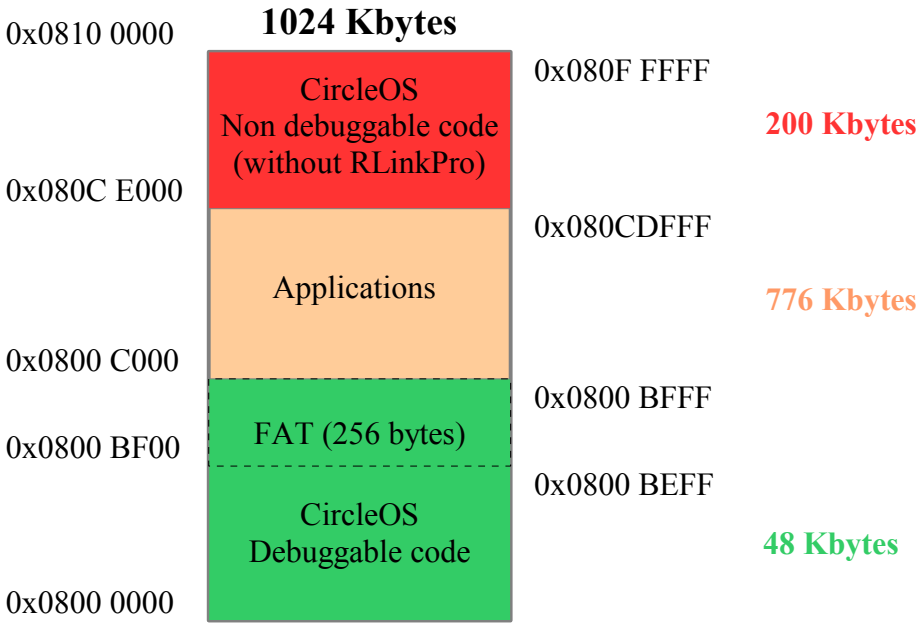
Open4 (STM32G):



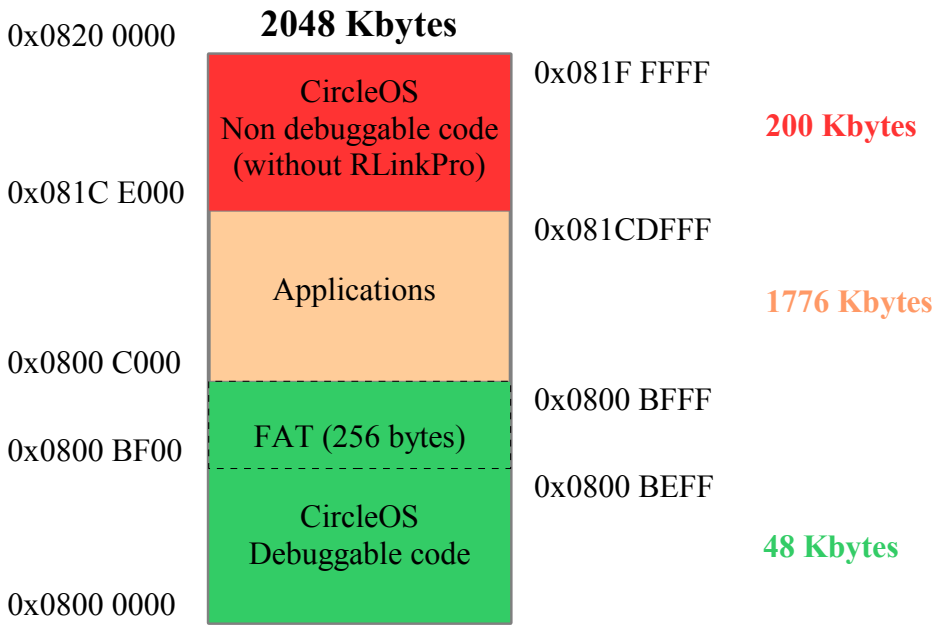
Open4 (STM32L):



Open4 (STM3240G):

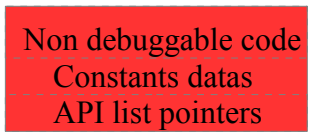


Open4 (STM3242x):



CircleOS non debbugable:

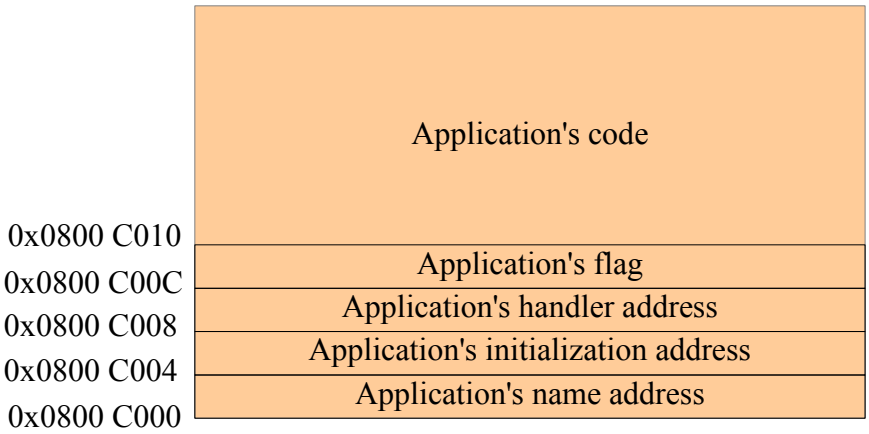
This is composed of a part of the CircleOS code, constant datas (pictures, sounds...) and the API list pointers.
This area is located at 0x801C000 (Primer 1), and 0x8062000 (Primer 2) for example).



Applications:

The applications are loaded here, the default length of an application is 8 KBytes for Primers1 & 2 and 16 KBytes for Open4 (due to the debug limitation) but can be changed, by modification of the link script (**circle_app.ld** or **Circle_App_OP4.ld**).

The standard shape of an application is:

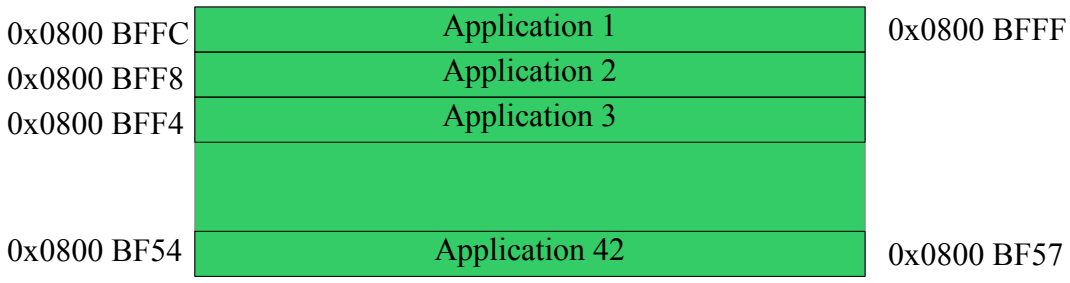


Here the address 0x0800 C000 is an example of the first application in an Open4.

FAT:

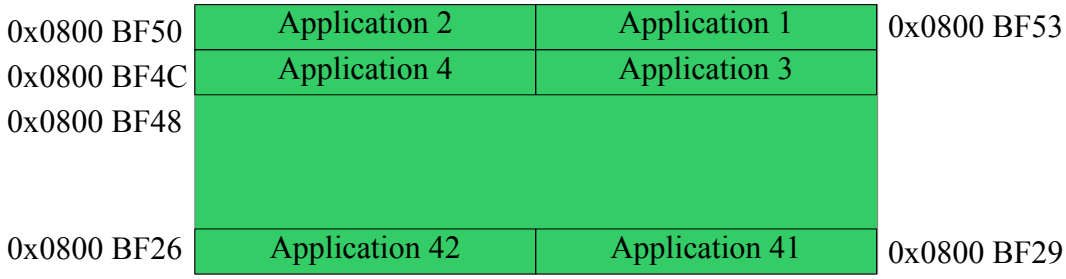
This contains a loaded applications list (6 bytes per applications / 42 applications max):

First area :



The 4 bytes indicates the application address.

Second area :



The 2 bytes indicates the application size in Kb (up to 64 Mb !).

CircleOS debbugable code :

Specific address:

0x08000108 : @ Circle OS version
0x08000104 : @ FAT address
0x08000100 : @ Circle API jump table address
0x08000FFE : @ Max FLASH available for OS + applications
0x08000FFC : @ Max RAM available for applications

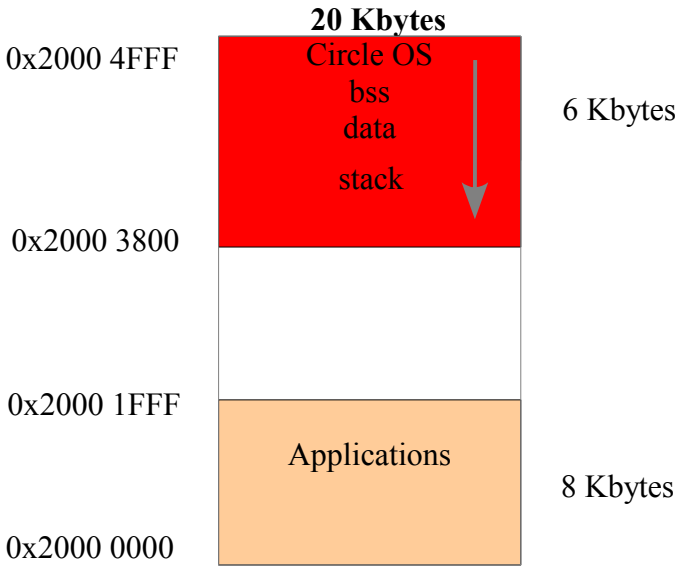
These informations are used by the Circle_Mgr tool, which allows to add, remove or list the applications.

2.2.2 RAM

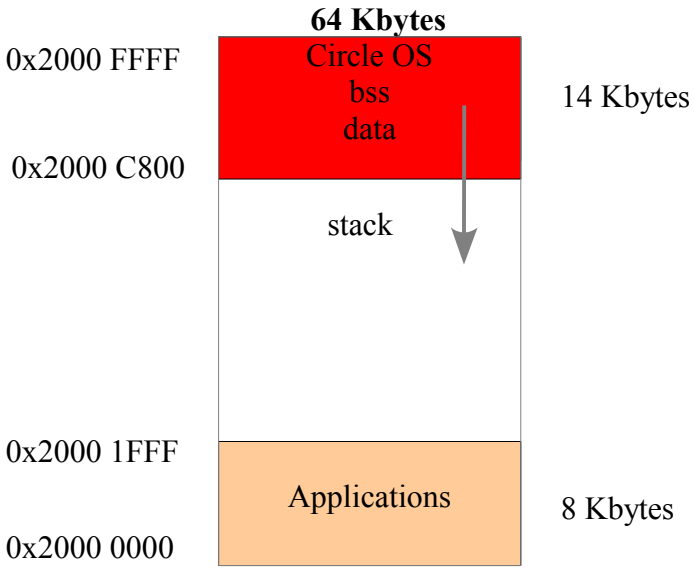
The standard size of an application workspace in the RAM is 16 Kbytes.

To increase the available RM modify the file "***Circle_App.ld"*** for Primer1 or Primer2, and "***Circle_App_OP4.ld***" for Open4/Evo Primer platforms.

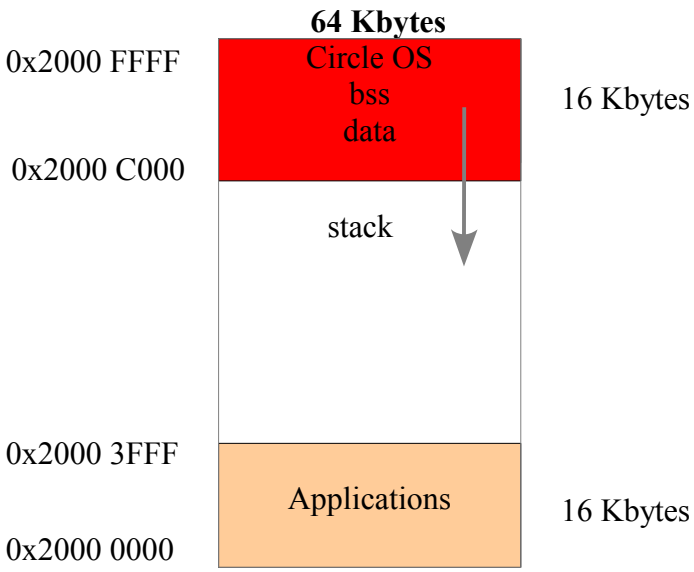
Primer 1:



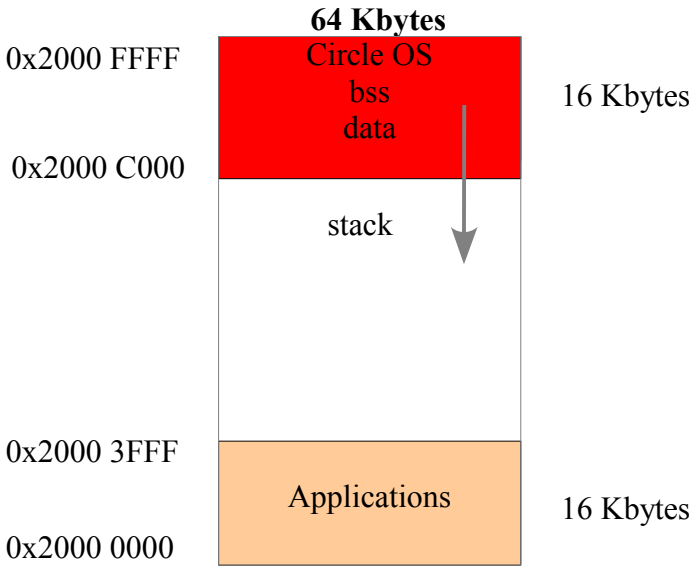
Primer 2:



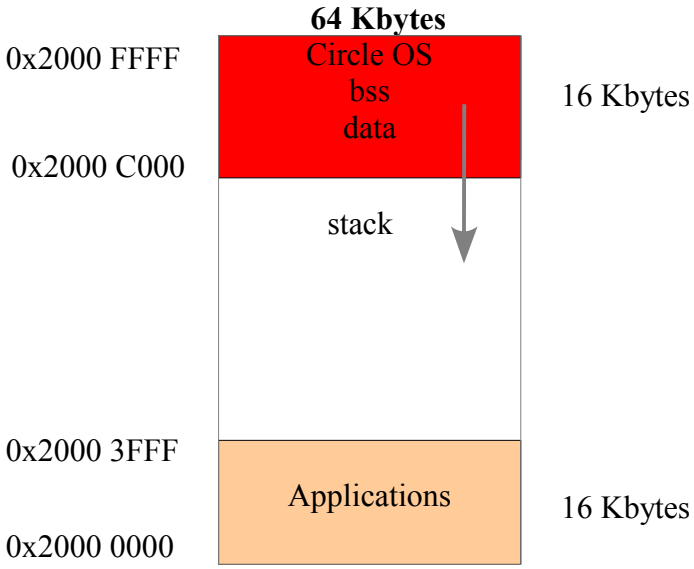
Open4 (STM32C):



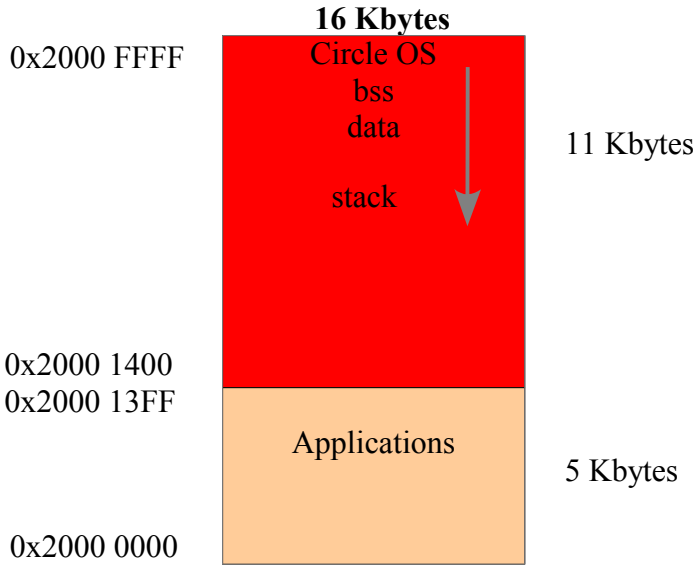
Open4 (STM32E):



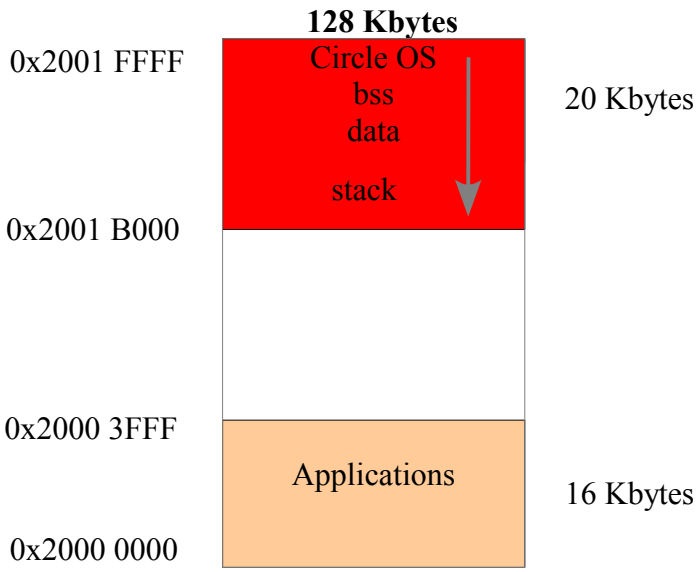
Open4 (STM32G):



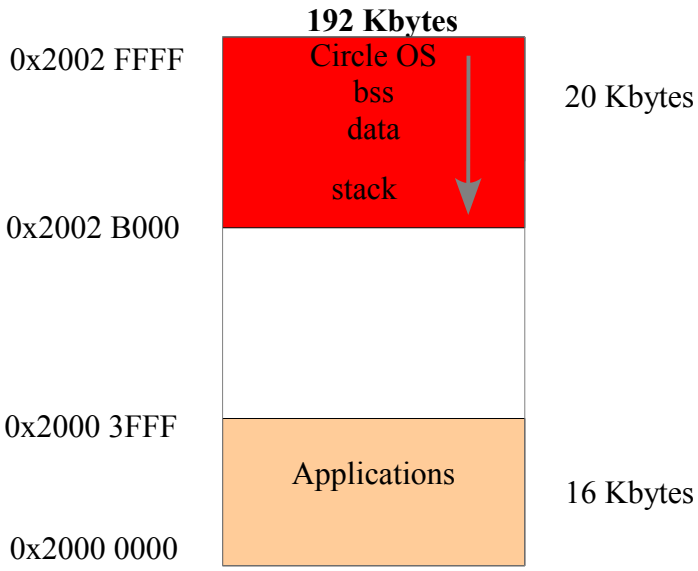
Open4 (STM32L):



Open4 (STM3240G):



Open4 (STM3242x):



2.3 Memory (STM8)

2.3.1 ROM

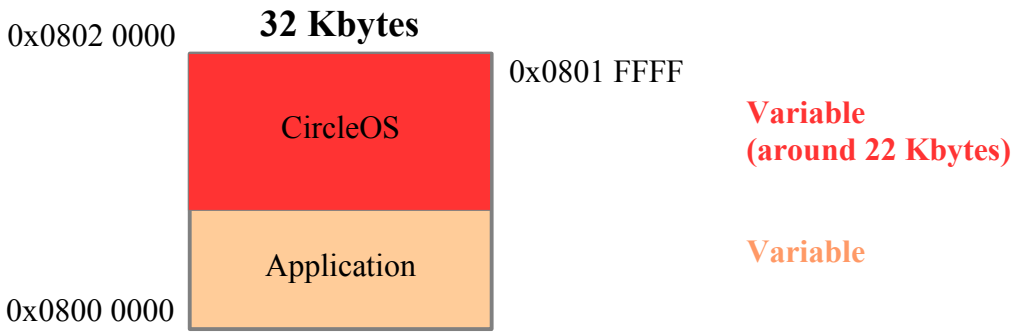
The ROM is divided between the flash memory, an internal EEPROM and an external EEPROM (called I2C EEPROM). The I2C EEPROM is used to store applications while the internal EEPROM contains the FAT part and the backup registers.

The application shares the flash memory with the CircleOS, as the OS takes around 26 Kbytes of memory, the application has 6 Kbytes available.

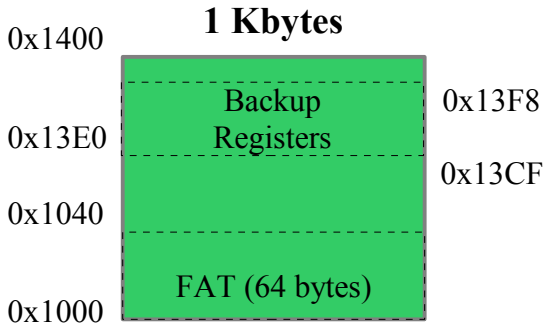
Applications are loaded in the I2C EEPROM with the Circle_STM8_mgr.exe. When a new application is used, the flash memory's 32Kbytes is substituted with one of the 32Kbytes blocs of the I2C EEPROM.

Note: You must be careful in the way you develop CircleOS applications for STM8L, so that your code does not consume too much ROM space. For instance, using “float” or “long” variables will require usage of expensive C runtime functions, which will consume a lot of space.

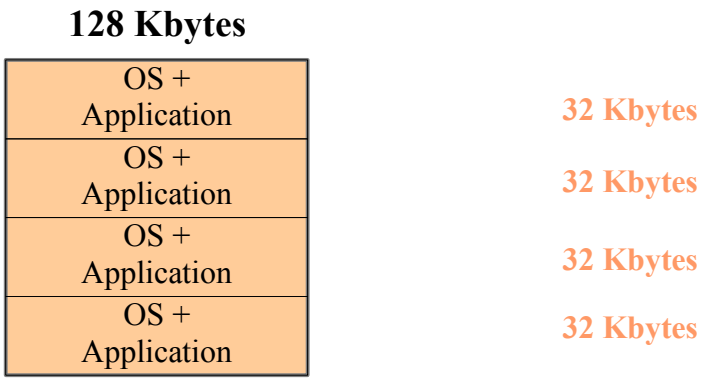
FLASH :



EEPROM :

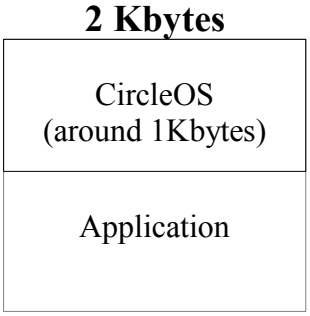


I2C EEPROM:



2.3.2 RAM

In the same way as ROM space, your application shares the RAM space with the CircleOS run-time. The CircleOS RAM usage is around 1KB, so you have 1KB available for your application.



2.4 Project files organization

Ride7 project :

While using Ride7 the project files are sorted by function (which is not the real sort on the hard disk):

- OS
- GUI
- Libraries
- Peripherals
- …

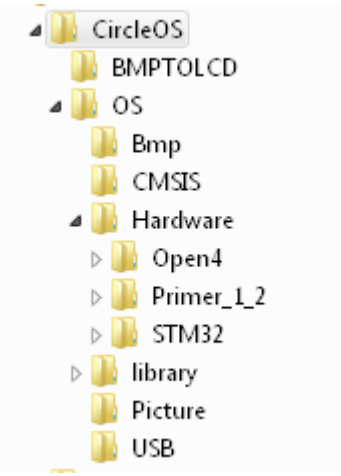
In each section files which are common between the processors are found in the root, the specific files can be found in the “hardware specific” sub-section.

Hard disk's files :

On the hard disk, all the common files (between both of the processors and the operating systems) are in the root folder. The specific files are in the “Hardware” folder, they are sorted by processors and OS.

Note: There are specific files for a family (for example: STM8) but there can be also specific files for a model in particular (for example: STM8L)

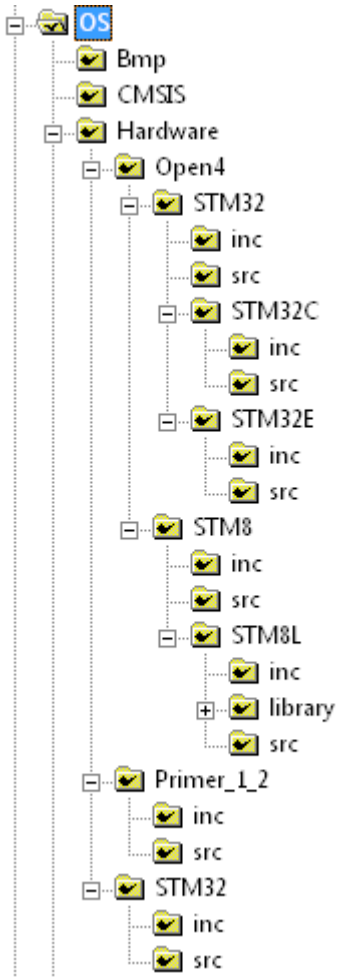
Global folders organization :



Folder contents :

- CircleOS : Ride7 project files for all platforms,
- CircleOS\OS : common files, shared with all platforms,
- CircleOS\library : STM32F peripherals library for Primer 1&2 and Open4 STM32F family,
- CircleOS\USB : STM32F USB library for Primer 1&2 and Open4 STM32F family,
- CircleOS\OS\Bmp : image header converted files,
- CircleOS\BMPTOLCD : image conversion tools,
- CircleOS\OS\Picture : image source files.

Hardware folders organization details :



Folder contents:

- Hardware\STM32 : common files, shared with STM32 platforms (Primers and Open4),
- Hardware\Open4\STM32 : common files, shared with STM32 Open4 platforms (Open4 only),
- Hardware\Primer_1_2 : specific files for Primer 1 and Primer2 platforms,
- Hardware\Open4\STM32x : specific files for one SM32 Open4 platform,
- Hardware\Open4\STM8 : common files, shared with STM8 Open4 platforms,
- Hardware\Open4\STM8x : specific files for one SM8 Open4 platform.

2.5 Software configuration

2.5.1 Shared declarations

All common declaration for internal use, are gathered in the “circle.h” header file.
This file is included into each CircleOS file.
The “circle_api.h” is the same header file for applications.

2.5.2 Specific declarations

All platform specific declaration are located in the “circle_platform.h” header file.
In particular, all hardware declarations are made here (peripheral ports, pin-out..).

Here are also configured the presence or not of the peripheral, thanks to macro, that allowos conditional compilation of common source parts :

Macro	Peripheral	Use
JOYSTICK_AVAIL	4 direction joystick	1 : Joystick is managed 0 : No joystick
LED_INV	Base LED's	1 : LED's are active at level 0 0 : LED's are active at level 1
POWER_MNGT	Battery charger circuitry	1 : Battery charger present 0 : Battery changer not present
SDCARD_AVAIL	SDCard	1 : SDCard is managed 0 : SDCard is not managed
TOUCHSCREEN_AVAIL	Touch screen	1 : Touch screen is managed 0 : Touch screen not available or not managed
AUDIO_AVAIL	Audio codec	1 : A codec is used or simulated 0 : Only buzzer
MEMS_POINTER	Mems	1 : MEMS present 0 : NO MEMS available
BACKLIGHT_INTERFACE	LCD	1 : The backlight level is managed 0 : The backlight level is fixed
MEMS_SPEC_MNGT	Mems	1 : The MEMS handler is called by special timer 0 : The MEMS handler is normally scheduled
MEMS_FULLINFO	Mems	1 : The MEMS handler computes all filters 0 : The MEMS handler provides few informations

Macro	Peripheral	Use
SDCARD_SDIO	SDCard	1 : SDCard is managed by SDIO peripheral 0 : SDCard is managed by SPI
DISPLAY_TEMP	Internal CPU temperature sensor	1 : Internal CPU temperature is displayed 0 : Internal CPU temperature not available or not displayed
EXT_FONT	LCD	1 : Extended font management (several system fonts) 0 : Only one system font
BUZZER_VOL_CONF	Buzzer	1 : Buzzer volume configurable 0 : Buzzer not available or not managed
CX_AVAIL	Cx	1 : Extension connector is managed 0 : Extension connector not available or not managed
DMA2D_AVAIL	DAM2D (Chrom-ART Accelerator)	1 : DAM2D is managed 0 : DAM2D not available or not managed

3. LCD

3.1 Files

Lcd.c
lcd_spe.c
font_spe.c
font_spe.h

3.2 Principles

3.2.1 Fonts

CircleOS provides several fonts, also application can import their own fonts. CircleOS default font is “Primer Font”, on the Open4 platform (except with the STM8L processor).
A second font “Medium Font” is available, bigger and easier to read on the Open4 screen, and a “Numbers” font is also available to display big numbers.

A font is defined by an array (type : u8*) that describes the pixels of each characters of the ASCII.
The format of characters is described in the following paragraph.

For font managing CircleOS needs information that must be stored in a font definition structure (type: tFontDef).
This structure hold :

- the font ID (which type is “*enum ENUM_FontID*”),
- the font width (max =16),
- the font height (max = 16),
- the default magnification coefficient,
- the first ASCII code of the font (typically 32, as codes which are under aren't characters),
- the last ASCII code of the font (255 if all the ASCII codes are provided),
- the font pointer to the data,
- the font name.

At this point you can add your personal font by using LCD_SetFont() with your own pointer to your font and LCD_SetFontDef() with the associated font definition.

In order to use the CircleOS fonts, a simple function is provided : LCD_ChangeFont(). The only parameters is the font ID (for example “0” or “FONT_PRIMER” for the default font). This feature is possible thanks to a table that indexes the font definitions.
This table is a tFontTable structure and only contains:

- the number of fonts
- the font definitions table (type : tFontDef*)

3.2.2 Character fonts

Fonts can be added but sizes are restricted :

- maximum width : 16 pixels,
- maximum height : 16 pixels.

When displayed a magnification coefficient can be applied to multiply the height and the width.
The ASCII characters are usually describe from 32 (space) to 255 (NULL) by a list of bytes.
Each bit is associated to a pixel.
The reading way is from the bottom to the top then from the left to the right.
So, if the font height is above 8, two consecutive bytes are used to display one column.

Example : character I (ACSII 73) :

Bit 0							
Bit 1							
Bit 2							
Bit 3							
Bit 4							
Bit 5		■	■	■	■	■	
Bit 6				■			
Bit 7				■			
Bytes	1 (0x00)	3 (0x20)	5 (0x20)	7 (0xe0)	9 (0x20)	11(0x20)	13(0x00)
Bit 0				■			
Bit 1				■			
Bit 2				■			
Bit 3				■			
Bit 4				■			
Bit 5		■	■	■	■	■	
Bit 6							
Bit 7							
Bytes	0 (0x00)	2 (0x10)	4 (0x10)	6 (0x1f)	8 0x10)	10(0x10)	12(0x00)

In the font_spe.h, this 14 bytes are stored as one row:
/* ASCII 73 */ 0x00, 0x00, 0x10, 0x20, 0x10, 0x20, 0x1f, 0xe0, 0x10, 0x20, 0x10, 0x20, 0x00, 0x00

3.2.3 Images

CircleOS can display image with the DRAW_SetImage() function.
Several format can be use with this function :

- Raw image (format used with previous CircleOS versions),
- 16-bits BMP image,
- 256 colours BMP image,

- 256 colours BMP image with RLE compression.

When called, the function search the information in the BMP header : a BMP always start with the code “0x42 0x4d” at the begin of the file, also the bytes number 6 to 9 should be reserved and set to 0. If those 2 conditions are fitted the data are read as BMP.

3.2.3.1 Raw format

Raw images only consist of 16-bit values, the image is printed like fonts from the bottom to the top and from the left to the right.

The 16-bit values consist of 5 bits for the red and blue colours and 6 bits for the green one.

Those bits are organized as followed :

Note: Colour bits organization : **GREEN(3 low bits) | BLUE(5) | RED(5) | GREEN(3 high bits)**
(G2G1G0B4 B3B2B1B0 R4R3R2R1 R0G5G4G3)

Standard values :

- Black = 0x0000,
- White = 0xFFFF,
- Blue = 0x1F00,
- Green = 0xE007,
- Red = 0x00F8.

Available tools can convert BMP image to header :

- BMPconverter.exe : convert any of the 3 BMP format image to a header file.
- image2primer.exe : convert any of image formats to raw image header.

3.2.3.2 BMP format

The BMP (BitMaP) format is a simple and common way to store images. Pixels are often wrote directly with theirs RGB codes. Also an RLE compression can be easily implemented.

The Circle OS uses 3 of the BMP methods :

- 16 bits colour depth,
- 256 palletised colours,
- 256 palletised colours with RLE compression.

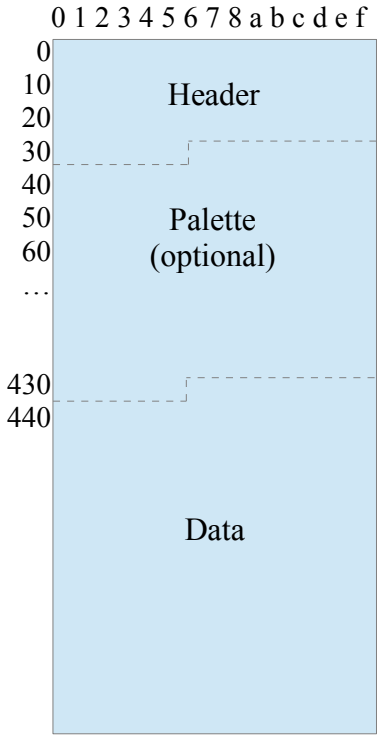
Note: With the “paint” windows' software, you can read BMP images with those parameters, but you can only wrote 256 palletised colours images. To create images in those formats more advanced software is needed (like : Gimp, Paint Shop Pro, Photoshop...).

A BMP file has a simple structure :

→ A header with the file information :

(Sizes are in bytes)

Offset	Size	Purpose
0x0	2	ID number of format (here 0x42 0x4d)
0x2	4	Size of the BMP file in bytes
0x6	4	Reserved
0xA	4	Offset of the data (usually 0x36 for a 16 bits image and 0x436 for a 256 colours palletised image)
0xE	4	Size of the header (from this point)
0x12	4	Bitmap Width in pixels
0x16	4	Bitmap Height in pixels
0x1A	2	Number of colour planes (usually 1)
0x1C	2	Number of bits per pixel
0x1E	4	Compression method (0 is used for no compression and 1 for 8 bits RLE)
0x22	4	Size of the data
0x26	4	Horizontal resolution (pixel per meter)
0x2A	4	Vertical resolution (pixel per meter)
0x2E	4	Number of colours in the palette (0 is used when there are 2^n colours)
0x32	4	Number of important colours



→ A palette :

There are 4 channels in the palette, this means that each colour is described with 4 bytes. The channels are the common Red, Green and Blue, but there is a fourth reserved channel (set to 0)

Note: In the palette, colours are not written with the RGB order but with the opposite. So each colours is described like : **BLUE** | **GREEN** | **RED** | **Reserved** .

→ The data :

This part is simply composed of the colour values. If the palette is used each colour is called by a single byte (instead of 2 for the 16 bit colour depth), which referred to the colour number in the palette.

Note: In the BMP specification, the width (in bits) of each line (of the image) has to be a multiple of 4. If not, the value 0 is written to complete the line. The CircleOS used only 8 or 16 bits values, so this problem can't occurred.

The RLE compression :

Instead of getting each values one by one, this method consist of getting 2 values : the first is a count of the second. Thus instead of writing “0x5A 0x5A 0x5A 0x5A 0x5A 0x5A”, we write “0x06 0x5A”.

The BMP method adds a way to prevent the problem of a streak of single values like “0xC2 0x14 0xD3 0x99” which has to become “0x01 0xC2 0x01 0x14 0x01 0xD3 0x01 0x99”. When the first value is 0x0,0 the second is the length of the following streak value. So “0xC2 0x14 0xD3 0x99” will become in a BMP file “0x00 0x04 0xC2 0x14 0xD3 0x99”.

Note: In BMP files as values are get 2 per 2, during the RLE decoding process the data pointer could point words instead of bytes (which is better to index large images). Because of value streak an offset can occur. So in BMP file, when the length of a streak is an odd number, an additional 0 is put (but has to not be confused with the colour 0 which is usually the black colour).

When a line end is reached the code “0x00 0x00” is written. This code can be used to avoid writing blank pixels at each end of line.
The “0x00 0x01” code is the end-of-file BMP code (can't occur in the data because a single value is written as “0x01 value”).

3.3 Functions

3.3.1 Internals Functions

3.3.2 APIs

`LCD_SetFontDef( tFontDef* FontDef )`

Change the current font definition (CurrentFontDef).

`LCD_ChangeFont( enum ENUM_FontID ID )`

Change both of CurrentFontDef and CurrentFont. This function is an easy way to change font between the CircleOS fonts (can change to an imported font). An application only has to call this function with the wanted font ID (FONT_PRIMER, FONT_PRIMERX2, FONT_MEDIUM...). The font IDs can be found in the circle_api.h file.

`LCD_GetTransparency()`

Get the transparency value for the characters. Return the value (type u8) of the transparency.

`LCD_SetTransparency( u8 NewTransparency )`

Get the transparency value for the characters. For now if the value is not zero the transparency is enable, but the use of the u8 type has been chosen to allow a shaded transparency (could be implemented in the future).

3.4 Structures

`tFontDef`

Store the font characteristics and their addresses.

- *ID* — Type enum ENUM_FontID
  This enum type is used for easy font changing. The programmers can use either the font number or is name (FONT_PRIMER, FONT_MEDIUM or FONT_NUMBERS...).
- *width*
- *height*
- *FontCoeff* — Font magnification coefficient
- *ASCII_start* — Used to know the begin of the font, for example the font FONT_NUMBERS can be used only to draw numbers : thus its ASCII_start is 48. Any ASCII code that are under this number won't be draw, according to the *LCD_DisplayChar()* ASCII code check
- *ASCII_end* — Used as ASCII_start for knowing the end of the font. The font FONT_NUMBERS end at the ASCII code 57
- *font* — Pointer of the font data (type u8*)
- *title* — String that contain the font name (could be used when adding new fonts or drawing current font information)

`tFontTable`

Structures used to index the fonts.

- *nb* — Number of fonts
- *fonts* — tFontDef list (type tFontDef*).
  Set to provide for programmers an easy access to fonts with their number or ID (see below). Indeed any CircleOS font can be access through the Font_Table variable (type tFontTable) with the syntax : Font_Table.fonts[FONT_ID]
  (FONT_ID can be for example FONT_PRIMER)

4. Audio

4.1 File

audio_spe.c

4.2 Principles

4.2.1 Generalities

The audio capabilities of the Primer2 are due to the codec STw5094A.

Two principal modes of the codec are used:

- voice mode, for recording,
- audio mode, for playing pre-recorded sounds. This is the default mode.
  In this mode, the inbuilt tone generator is used as a buzzer.

4.2.2 I2S implementation

Audio data is exchanged over two I2S buses:

- I2S2 : connected to the SPI2 entry of the STM32, used in voice mode to record sounds from the microphone.
- I2S3 : connected to the SPI3 output of the STM32, used in audio mode to play sounds from memory.

The two buses are managed by interrupts, and two functions :

- `AUDIO_Play` : initializes pointers and indexes, enables IT; then the transfer is done under interrupt until the end of the buffer is reached, and while the audio status "`IS_PLAYING`" is set.
- `AUDIO_Record` : initializes pointers and indexes , enables IT; then the transfer is done under interrupt until the end of the buffer is reached, and while the audio status "`IS_RECORDING`" is set.

See the STw5094A datasheet for more information.

4.2.3 I2C implementation

A 22 value table "`AUDIO_CODEC_CRs`" keeps an image of all the codec registers' values.
When an I2C write is requested, `AUDIO_CODEC_CRs` is updated with the new values, then a flag "`flagWrite_AUDIO_CODEC_CRs` " is set up to the number of registers to write. This flag is detected by the AUDIO Handler :

- if the value is between 0 and 21, the register with this address value is written,
- otherwise the whole 22 registers are written.

The macro used to set the flag is `SET_FLAG_WRITE_CODEC_CRS(x)`.

4.3 Functions

4.3.1 Internal functions

`AUDIO_Init()`

Performs general initialization of the STw5094A audio codec. Only the I2C interface is activated.

- Set the clocks,
- Restore backup settings or set the default values,
- Initialize the I2C bus,
- Reset the codec,
- Get the default codec configuration from the codec,
- Init the default audio mode to play 16 bits / 8 kHz audio sounds.

`AUDIO_Handler()`

Called by the CircleOS scheduler to manage audio tasks.

- Apply configuration change if necessary (speaker on/off set by menu)
- Disable interrupts, if requested, according to the `AUDIO_Recording_status` or `AUDIO_Playing_status`,

`AUDIO_I2C_Init()`

Configures GPIO, ClockSpeed = 100 kHz.

`AUDIO_Init_Audio(length, frequence, format)`

Initializes the codec in audio mode.

- Set the codec configuration : mute, volume, tone frequency,
- Initialize the I2S bus (SPI3),
- parameters :
  - length : 16, 24 or 32 bits. Required: Convert 8-bit parameter to 16-bit,
  - frequency : 8 kHz or 16 kHz,
  - format : mono or stereo.

`AUDIO_Init_Voice(length, frequence, format)`

Initializes the codec in voice mode.

- Set the codec configuration : mute, volume, tone frequency,
- Initialize the I2S bus (SPI2),
- parameters :
  - length : 16, 24 or 32 bits. Required: Convert 8-bit parameter to 16-bit,
  - frequency : 8 kHz or 16 kHz,
  - format : mono or stereo.

`AUDIO_Shutdown()`

Stops the IT on SPI bus, resets the codec. Should be called before shutting down the Primer, to avoid electrical problems.

`AUDIO_Welcome_Msg()`

Plays the welcome pre-recorded message, launched during the power on phase.

AUDIO_I2C_Read_Register(register to read)

Calls the AUDIO_I2C_ReadMultByte function, with parameter 1 register to read.

AUDIO_I2C_Write_register(register number)

Calls the AUDIO_I2C_WriteMultiByte function with number =1;

AUDIO_I2C_ReadMultiByte(1er register number, number of registers, @ reception buffer)

Reads x registers of the codec through the I2C bus, with polling method.

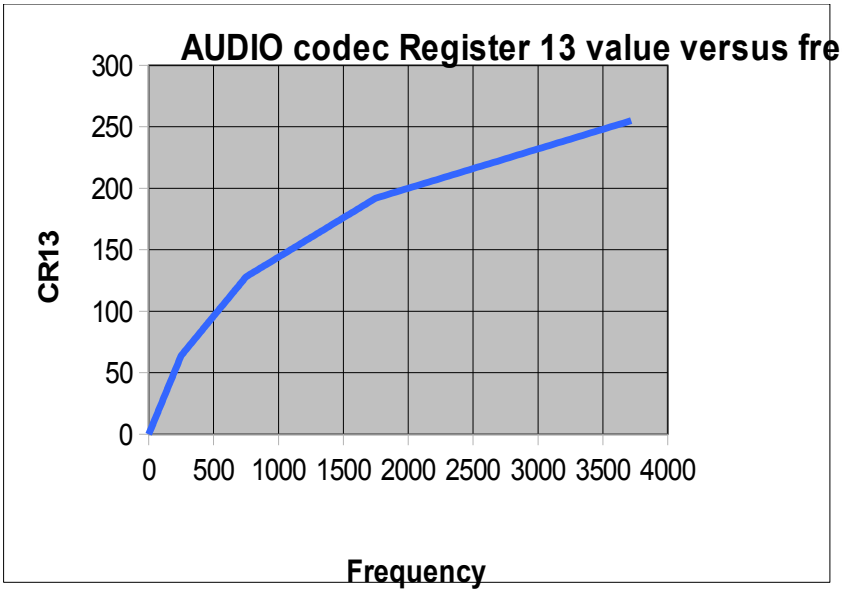
AUDIO_I2C_WriteMultiByte(1er register number, number of registers, @ emission buffer)

Writes x registers of the codec through the I2C bus, with polling method.

AUDIO_DeviceSoftwareReset

AUDIO_BUZZER_SetToneFrequency(frequency)

Sets the frequency of the tone generator, when playing buzzer sounds.
The frequency value accepted is 0 to 3800 Hz. This function converts the frequency to register value O to 255 according to the table provided by the STw5094A datasheet. See graph below.



AUDIO_Set_Volume()

Applies the "AUDIO_Volume" gain value to the codec, for the loudspeaker (steps from +6 to -24 dB). The volume value is divided by 2 before it is applied to the headphone (attenuation multiplied by 2, steps from +0 to -40 dB).

AUDIO_Cpy_Mono()

Copy Mono data to small buffer, set both Left+Right Channel to same value

4.3.2 APIs

AUDIO_SetMode(new mode)

Changes the codec mode.

AUDIO_GetMode()

Gets the current codec mode.

AUDIO_Play(buffer, size)

Issues audio samples (stored in buffer) to the audio codec via I2S.

AUDIO_Playback_Stop()

IStop the playback by stopping the DMA transfer.

AUDIO_Record(buffer, size)

Stores audio samples in the buffer from the audio codec via I2S.

AUDIO_Record_Stop

Stop the record by stopping the DMA transfer.

AUDIO_Playback_GetStatus()

Gets the status of playback mode (`NO_SOUND`, `IS_PLAYING`).
Required : to check before a new call to `AUDIO_Play`, or `AUDIO_Record`.

AUDIO_PlaybackBuffer_GetStatus(value)

Gets the status of Playback buffer.

AUDIO_Record_Buffer_GetStatus(value)

Gets the status of Record buffer.

AUDIO_Recording_GetStatus()

Gets the status of recording mode (`NO_RECORD`, `IS_RECORDING`).
Required : to check before a new call to `AUDIO_Play`, or `AUDIO_Record`.

AUDIO_SPEAKER_OnOff(ON/OFF)

Sets the PLS switch of audio codec ON or OFF, to mute or not the loudspeaker.
If PLS = ON, the loudspeaker is active for audio and buzzer.

AUDIO_MUTE(ON/ OFF)

Sets the MUT switch of audio codec ON or OFF.
If MUT = ON, buzzer, loudspeaker and headphones are all cut off.

AUDIO_IsMute()

Indicates if audio is MUTE or not. If MUT = ON, buzzer, loudspeaker and headphones are all cut off.

AUDIO_Inc_Volume(dB number)

Increases the general gain of number of dB.

AUDIO_Dec_Volume(dB number)

Decreases the general gain of number of dB.

AUDIO_ReadRegister(register_to_read)

Reads a data byte from one of STw5094A configuration registers.

AUDIO_ReadRegister(register_to_read, data_to_read)

Send a data byte to one of STw5094A configuration registers.

4.4 Structures

None

5. Touchscreen

5.1 Files

touchscreen.c
touchscreen_spe.c

5.2 Principle

The touch screen is connected to 4 analog inputs of the STM32 (ADC1 channels 10 to 13). The measurement is collected by DMA. Configuration and initialization of the DMA is made in the function "`ADConverter_Init`" (file "adc.c"), which is launched by the main function.

5.3 Functions

5.3.1 Internal functions

`TOUCHSCR_Init()`

Initializes the touchscreen handler :

- retrieve calibration informations from backup registers,
- position initial values,
- test if first power up (backup registers empty) and launch calibration if yes, drawing if not.

`TOUCHSCR_Handler()`

Calculates median value out of ADC_NB_SAMPLES samples,

- convert voltage to points,
- update of the touch position,
- correction,
- calibration coefficients application.

`TOUCHSCREEN_Calibration()`

Calibration sequence

- display text : "Press on the black cross with a stylus" and the first cross,
- wait for touch,
- memorize the positions,
- display the second cross,
- wait for touch,
- memorize the positions,
- same thing with the last cross
- calculate and store the coefficients (in the backup registers too).

`TOUCHSCR_SetMode()`

Changes the touchscreen mode. Available modes : NORMAL, DRAWING or CALIBRATION.

TOUCHSCREEN_Drawing()

Provides a mini "scribble" functionality, it is active when no applications are running.

MedianFilter()

Provides the median of the array of acquisition measures by DMA.

TOUCHSCR_CalculateCalibration()

Use the three acquired points Raw[] and the three reference points TS_RefSample[] to calculate and update the calibration matrix coefficients TS_CalibrationSetup[].

5.3.2 APIs

TOUCHSCR_GetPos()

Returns the current position of the point touched.
Format : X in the LSB and Y in the MSB.

Warning : the (0x0) points to the low left corner. The position depends on the current orientation of the screen.

TOUCHSCR_GetAbsPos()

Returns the absolute current position of the point touched.
Format : X in the LSB and Y in the MSB.

Warning : the (0x0) points to the low left corner. The position does NOT depends on the current orientation of the screen.

TOUCHSCR_IsPressed()

Indicates if the screen has been touched or not.

TOUCHSCR_GetMode()

Indicates if the touchscreen is in calibration or not.
Available modes : NORMAL, DRAWING or CALIBRATION, if pending.

TOUCHSCR_SetSensibility()

Modifies the touch detection sensitivity.
Value 0 to 4095 (3000 by default).

5.4 Structures

TOUCHSCR_Info

Mode and coordinates of the last touched point

- xRaw (non calibrated coordinates)
- yRaw
- xAbsPos (absolute coordinates)
- yAbsPos

- xPos (orientation dependant coordinates)
- yPos
- TouchPress (boolean)
- Mode (TS_NORMAL, TS_DRAWING, TS_CALIBRATION, TS_POINTER)

TOUCHSCR_Cal

Calibration coefficients (see section 4.5)

- A
- B
- C
- D
- E
- F
- R

tPOINT

- X
- Y

5.5 Calibration principle

5.5.1 Conversion

As the values obtained by the ADC (vX, vY and vT) are not coordinates, we need a conversion to obtain coordinates : (Ucc = 4096 is the maximum value given by the ADC)

X = vX * 1000 / (Ucc - vY)

Y = (vY - vT) * 1000 / (Ucc - vY)

T = (vT – vX) * 1000 / (Ucc - vY)

5.5.2 Correction

The coordinates obtained (X and Y) may not be true display coordinates(Xd and Yd), we have to correct those coordinates with coefficients (from A to F and R) calculated during the calibration.

The correction's formulas are :

Xd = (A*X + B*Y + C) / R

Yd = (D*X + E*Y + F) / R

Those coefficients are stored in the **TS_CalibrationSetup** structure.

5.5.3 Calibration

The calibration code is inspired from Carlos E. Vidales code (see article here “http://www.eetindia.co.in/STATIC/PDF/200206/EEIOL_2002JUN02_EMS_OPTO_TA.pdf?SOURCES=DOWNLOAD”, and code is available here “ftp://ftp.embedded.com/pub/2002/06vidales“.

To calculate the coefficients used for correcting coordinates we need, during a calibration, 3 points' coordinates. Their placements should always be far between each to obtain good results. In the CircleOS the points are at three corners with a 10% margin.

During the calibration step, we need to gather the uncalibrated coordinates of the 3 points (Xt0, Yt0, Xt1, ...) and their theoretical coordinates (Xd0, Yd0, Xd1, …), as we know their locations.

Then we can calculate the coefficients:

R = (Xt0 – Xt2) * (Yt1 – Yt2) – (Xt1 – Xt2) * (Yt0 – Yt2)

A = (Xd0 – Xd2) * (Yt1 – Yt2) – (Xd1 – Xd2) * (Yt0 – Yt2)

B = (Xt0 – Xt2) * (Xd1 – Xd2) – (Xd0 – Xd2) * (Xt1 – Xt2)

C = Yt0 * (Xt2 * Xd1 – Xt1 * Xd2) + Yt1 * (Xt0 * Xd2 – Xt2 * Xd0) + Yt2 * (Xt1 * Xd0 – Xt0 * Xd1)

D = (Yd0 – Yd2) * (Yt1 – Yt2) – (Yd1 – Yd2) * (Yt0 – Yt2)

E = (Xt0 – Xt2) * (Yd1 – Yd2) – (Yd0 – Yd2) * (Xt1 – Xt2)

F = Yt0 * (Xt2 * Yd1 – Xt1 * Yd2) + Yt1 * (Xt0 * Yd2 – Xt2 * Yd0) + Yt2 * (Xt1 * Yd0 – Xt0 * Yd1)

Then we save the coefficients into the TOUCHSCR_Cal structure, and we save the 3 real point coordinates into the backup registers.

5.5.4 Median filter

During each measurement the ADC returns several values, in order to have the best reliability we only take the median of the values list, which allows to reject the spike values.

6. Toolbar

6.1 Files

toolbar.c : common management
toolbar_spe.c : platform specific icons declaration

6.2 Functions

6.2.1 Internal functions

`TOOLBAR_Init()`

Sets the default toolbar, and first drawing of the icons.

`TOOLBAR_Handler()`

Waits for a touch on one of the icons, and launches the appropriate function.
Wait for a redraw request and draws the icons accordantly.

`TOOLBAR_Button(button, bgnd_color, bgnd_color_sel, sel)`

Draws the icon of a specified button.
If the item is selected, the colour designed by "`bgnd_color` " is replaced with the colour designed by "`bgnd_color_sel`".

`TOOLBAR_UnSelectButton(button)`

Draw the button with unselected colours.

`TOOLBAR_SelectButton(button)`

Draws the button with selected colours, sends a beep, and launches the programmed function.

`TOOLBAR_Redraw(request)`

Set the internal flag, check by the handler in order to redraw the buttons, accordantly to the request :

- bit 0 : button 0 to update,
- bit 1 : button 1 to update,
- bit 2 : button 2 to update,
- bit 3 : button 3 to update,
- bit 4 : set default toolbar.

Predefined requests :

- #define TOOLBAR_REDRAW_BUTTON0 0x01
- #define TOOLBAR_REDRAW_BUTTON1 0x02
- #define TOOLBAR_REDRAW_BUTTON2 0x04
- #define TOOLBAR_REDRAW_BUTTON3 0x08
- #define TOOLBAR_REDRAW 0x0F
- #define TOOLBAR_DEFAULT 0x10

Note : the request can be gathered, but the default toolbar request has priority to the other requests. So, if you set bits in addition to the bit 4 at the same time, this bits will not be taken into account.

`flowSound(), fHighSound(), fMuteSoundf()`

System functions corresponding to the actions of the 3 first buttons.

`DefaultAction()`

Action of the first button : launches the settings menu, or the last launched application.

6.2.2 APIs

`TOOLBAR_Set(@ new toolbar)`

Changes the current toolbar to the new one described by the structure passed through the parameter.

`TOOLBAR_SetDefaultToolbar`

Restores the system toolbar by default.

`TOOLBAR_ChangeButton(button, @newicon, @function)`

Replaces the button with a new one (new icon and new function).

6.3 Structures

`tToolbar()`

Configuration information of the toolbar:

- `.nbItems` : 0 to `TOOLBAR_MAXITEMS`
- `.FirstDispItem` : index of the first displayed item
- `.tToolbar_Items` : array of `TOOLBAR_MAXITEMS` `tToolbar_Items`

`tToolbar_Item()`

- @ of the icon item (32 x 32 pixels RLE bmp format for Primer2, 60 x 60 pixels RLE bmp format for Open4),
- @ of manage function : to be launched when the user “click” on the corresponding button,
- `TOOLBAR_MAXITEMS` = 4 for the moment.

7. Button / Joystick

7.1 Files

button.c
button_spe.c

7.2 Functions

7.2.1 Internal functions

`JOYSTICK_Handler()`

Called by the button handler, calls `GetNewState` and `WaitForRealease`, and manages the anti-bouncing filter.

`JOYSTICK_CircularPermutation(abs_direction, iter)`

Permutes clockwise the joystick state :

left -> up -> right -> down -> left.

left_up-> right_up -> right_down -> left_down -> left_up.

7.2.2 APIs

`BUTTON_SetMode(mode)`

Set new button mode. Possible mode :

- BUTTON_DISABLED = -1,
- BUTTON_ONOFF = 0,
- BUTTON_ONOFF_FORMAIN = 1 (catch by the CircleOS if = 1),
- BUTTON_WITHCLICK = 2

`BUTTON_GetMode()`

Returns the mode.

`BUTTON_GetState()`

Returns the state. Possible states :

- BUTTON_UNDEF = -1,
- BUTTON_RELEASED = 0,
- BUTTON_PUSHED = 1,
- BUTTON_PUSHED_FORMAIN = 2 ,
- BUTTON_CLICK = 3,
- BUTTON_DBLCLICK = 4

`BUTTON_WaitForRelease()`

Disable temporarily any new button event for the anti-bouncing filter.

`JOYSTICK_GetState()`

Returns the state. Possible states :

- JOYSTICK_UNDEF = -1,
- JOYSTICK_RELEASED = 0,
- JOYSTICK_LEFT = 1,
- JOYSTICK_RIGHT = 2,
- JOYSTICK_UP = 3,
- JOYSTICK_DOWN = 4
- JOYSTICK_RIGHT_UP = 11, /*JOYSTICK_RIGHT | JOYSTICK_UP*/
- JOYSTICK_LEFT_UP = 5, /*JOYSTICK_LEFT | JOYSTICK_UP*/
- JOYSTICK_RIGHT_DOWN =13, /*JOYSTICK_RIGHT | JOYSTICK_DOWN*/
- JOYSTICK_LEFT_DOWN = 7, /*JOYSTICK_LEFT | JOYSTICK_DOWN */

JOYSTICK_WaitForRelease`()`

Disable temporarily any new button event for the anti-bouncing filter.

7.3 Structures

None

8. Power

8.1 File

shutdown.c

8.2 Power management principle

8.2.1 Power states

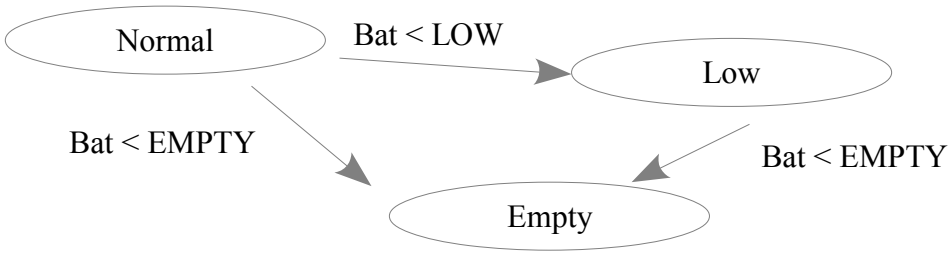
Possible states of the power management :

- PWR_STATE_UNDEF :do not know yet
- PWR_STATE_NOBAT : battery is not connected or dead
- PWR_STATE_CHARGING : power supplied from external source, battery is being charged
- PWR_STATE_FULL : power supplied from external source, charge is done (full)
- PWR_STATE_NORMAL : power from battery, normal level
- PWR_STATE_LOW : power from battery but battery is low (warning message displayed)
- PWR_STATE_EMPTY : power from battery but battery is critically low, shutdown to be processed.

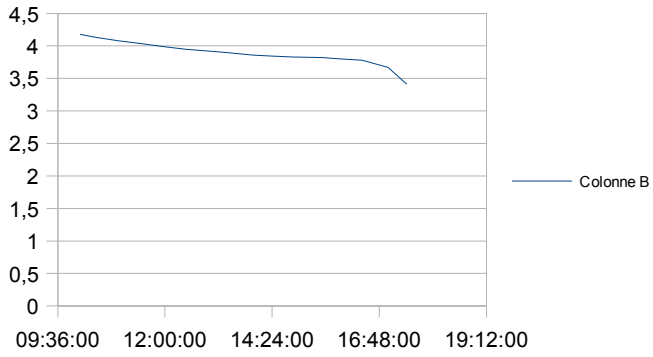
Thresholds :

- FULL = 4200 mV,
- LOW = 3750 mV,
- EMPTY = 3500 mV,
- NOBAT = 3000 mV.

The state FULL and CHARGING are provided by the battery charger circuit.
The other states are calculated with the battery voltage level :



The graph below shows battery voltage level versus time.



8.2.2 Standby management

In order to improve the battery life, the Primer is automatically powered off after a period of inactivity, if no application is running.

Two macros must be used to prevent shutdown :

- `PWR_SET_TIME` : sets the reference time, and launches the no-activity detection,
- `PWR_RESET_TIME` : resets the reference time, and thus stops the no-activity detection.

The delay before shutdown is defined by the macro `MAX_TIME_ON` : ~ 5 mn.

8.3 Functions

8.3.1 Internal functions

`POWER_Init()`

Initializes GPIO.

`POWER_Handler()`

- gets the battery voltage,
- gets the battery charger state,
- manages the power level,
- manages the standby function.

8.3.2 APIs

`SHUTDOWN_Action()`

- back-up system values,
- disables TIM2,
- stops the audio codec,
- powers off.

8.4 Structures

None

9. List

This section only concern platforms with touch screen (Primer 2 and Open4).

9.1 Files

list.c
list_spe.c

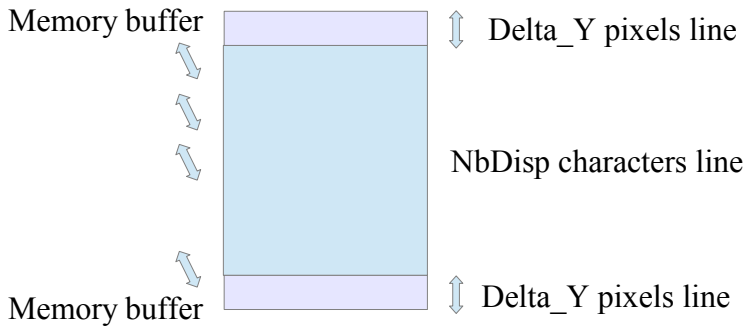
9.2 Principles

9.2.1 List

When the user moves his finger on the screen (like on an iPod), or when he uses the MEMs or the joystick, the list scrolls up or down depending on the interface:

- touch screen : moves until the new touch point,
- joystick or MEMs : moves one by one character.

The scroll is made DELTA_Y by DELTA_Y lines of pixels. Where DELTA_Y = 2 pixels.



During the scrolling 3 types of action occur (go to the bottom of the screen and top of the list in this example) :

1. **Scrolling LCD to LCD lines**: this phase uses the functions LIST_LCD_RectRead and LIST_DRAW_SetImage. These functions are derivatives of existing functions LCD_RectRead and DRAW_SetImage with the same aim (capture and saving of an LCD area), but they use DMA transfer, and 18-bit RGB colour mode (for Primer2).
2. **Scrolling from external memory to LCD first lines**: this phase uses cache memory where we write part of the characters lines, then we transfer the cache to the LCD.
   It uses the new function LIST_StoreString and the existing DRAW_SetImage.

When the scrolling is in the other sense (going to the top of the screen and bottom of the list), the phases are inverted, but the principle is identical.

Note : Primer2 uses DMA transfer between LCD and memory buffer. Unfortunately, the Open4's IL9325 LCD controller does not allow this transfer type.

9.3 Functions

9.3.1 Internal functions

LIST_GetNewSelectedItem()

Determines if and which item has been selected from the displayed list.
Returns the index of the selected item.

Formula : SelectedItem = ((Max_Y_List - Y)) / ListCharHeight, where:

ListCharHeight is the height of a character,
Y is the coordinate of the touch point,
Max_Y is the coordinate of the top of the list.

LIST_DetectMove()

Detects the direction of the move on the touchscreen.
Returns the number of moves (= number of "Delta y" between the new touch point and the previous touchpoint). Formula : dir = (move_cur_Y - move_old_Y) / DeltaY

- dir positive : go to the end of the list,
- dir negative : go to the top of the list.

LIST_RefreshItem(item selected, text mode)

Displays the item requested.

LIST_StoreString(@buffer, @string, length, offset, nblines, TextColor, BGndColor, CharMagniCoeff)

Stores `nblines` lines of characters into `buffer` for further display.

LIST_StoreChar(@buffer, @bmp, nblines, offset, TextColor, BGndColor, CharMagniCoeff)

Stores into buffer the provided ASCII character with the provided text (contained in bmp), background colours and the provided magnification coefficient. offset indicates the line number where the transfer should begin.

- 2 bytes by pixel.
- Called by LIST_StoreString.

InitListDMA() (Primer 2 only)

Prepares the DMA for transfer before saving or drawing an image. This function must be called before each transfer (one shot transfer).

LIST_LCD_RectRead(x, y, width, height)

Saves the pixels of a rectangle part of the LCD into a buffer.

- We get the image in 18-bit RGB colour mode (6 bits x 3) for Primer2, but in 16-bit RGB colour mode for Open4.
- The transfer is made by DMA (Primer 2), the bitmap is saved into the buffer `bmpTmp`.
- DMA transfer must be in 16-bit format due to the 4-bit shift of bus between STM32 and LCD.
- Then the image size in memory is 3 x 8 bits x size of the area to save for Primer2, and 2 x 8 bits x size of the area to save for Open4.

LIST_DRAW_SetImage(x, y, width, height)

Draws a color bitmap at the provided coordinates.

- The bitmap is made width height 2 byte words.
- Each 2 byte word contains the RGB colour of a pixel.
- The image is sent in 18-bit RGB colour mode (6 bits x 3) for Primer2, but in 16-bit RGB colour mode for Open4.
- The transfer is made by DMA (Primer2), the bitmap is saved into the buffer `bmpTmp`.
- For Primer 2, DMA transfer must be in 16-bit format due to the 4-bit shift of bus between STM32 and LCD.

9.3.2 APIs

`LIST_Set(@ list, posX, posY, center)`

Displays the list stated by the parameter, at the indicated position. If the boolean `center` is set, the list is centred on the screen, and `posX`/`posY` are ignored. `posX`, `posY` are the coordinates of the top left corner of the list.

`LIST_Manager()`

Manages the scroll of the list, with MEMS, joystick or touch screen.
Returns the index of the item selected (double click with the MEMs), if not, it returns -1.

9.4 Structures

`tList()`

List configuration information :

- .fdispTitle : presence of a title,
- .@ Title : title of the list
- .NbItems : 0 to LIST_MAXITEMS,
- .LgMax : maximimum number of character (calculated by the manager),
- .XPos, .Ypos : position of the list (updated by the manager),
- .XSize, .Ysize : size of the list (updated by the manager)
- .NbDisp : number of lines to be displayed at the same time,
- .SelectedItem : current selected item, (updated by the manager),
- .FirstDispItem : index of the first displayed item, (updated by the manager),
- .tListItem : array of LIST_MAXITEMS tListItems

`tListItem()`

- .Text : @ text of the item
- `LIST_MAXITEMS` = 64.

10. Menu

10.1 Files

menu.c
menu_settings.c
menu_spe.c
menu_app_spe.c

10.2 Principles

The menu handler is designed for monitoring the OS's menus and launching the applications.

Any menu is defined with the **tMenu** structure which is basically composed of:

- **fdispTitle** : The display option (boolean)
- **Title**
- **NbItems** : the number of items in this menu
- Other dynamic parameters (**Xpos**, **Xsize**...)
- **SelectedItem**
- **Items**[] : the list of the menu's items

Items are defined with the **tMenuItem** structure, it has:

- **Text**
- **Fct_Init**() : the initialization process
- **Fct_Manage**() : the main process
- **fMenuFlag** : a flag to provide some options

A standard menu's item doesn't have **Fct_Manage**(), also **Fct_Init**() only consist of opening the menu with the **MENU_Set(**) function (with the appropriate **tMenu** as parameter).

fmenuFlag can have two options tested by the 2 first bits.

- **REMOVE_MENU** : used to closed the former menu (used for commands like “QUIT” or “About”)
- **APP_MENU** : used to initialize the screen, the buttons, the pointer and close the menu (used for applications)

Any menu or application has to finish with a **MENU_code**.
Return codes for menus or commands' initialization :

- **MENU_CHANGE** : changes the current menu (in order to open a new menu).
- **MENU_LEAVE_AS_IT** : leaves the current menu without clearing the screen.
- **MENU_LEAVE** : leaves the current menu and comes back to the main menu if **REMOVE_MENU** is present.
- **MENU_REFRESH** : display the current menu.
- **MENU_CONTINUE** : let the current menu unchanged.
- **MENU_CONTINUE_COMMAND** : launch **Fct_Manage**() and restore the current command (if changed during the initialization).

Return codes for applications' initialization or main process of both applications and commands :

- **MENU_LEAVE** : leave the application (doesn't launch the **Fct_Manage**())
- **MENU_CONTINUE** : launch **Fct_Manage**() (for the initialization) or keep launching it (for the main process).

There is a special return code for application main functions (can be used also by commands) :

- **MENU_RESTORE_COMMAND** : restore the previous command (used to allow applications to open menus and leave them without leaving the application).

The menu handler works by following this steps (STOP means leaving the handler) :

Preliminary tests:

✔ If (the OS initialization isn't finished) :
  ➔ STOP
✔ If (there is no menu AND the button is pushed in the main screen) :
  ➔ opening of the main menu
✔ if (shut down's flag is on) :
  ➔ shutting down

Initialization:

✔ If (a current menu is present and there is no command running) :

  ✔ If (an item is selected) :

    ✔ if (the current menu is the application menu AND the item isn't “QUIT”) :
      ➔ storing the selected item in the backup register (last application launched)
    ➔ current command updated with the selected item

    ✔ If (command's flag has “REMOVE_MENU” bit) :
      ➔ closing the menus to the main screen

    ✔ if (the current menu has items) :
      ✔ if (command's flag has “APP_MENU” bit) :
        ➔ initializing the screen, pointer and button
      ➔ launching **Fct_Init**()

      ✔ if (return code isn't “MENU_CHANGE” nor “MENU_CONTINUE_COMMAND)” :
        ➔ retrieving the former menu for the current menu
    ✔ else :
      ➔ STOP

    ✔ switch (return code of **Fct_Init**()) :
      ➔ MENU_LEAVE_AS_IT : current command and menu to 0
      ➔ MENU_LEAVE : current command and menu to 0 and leaving menus to the main screen
      ➔ MENU_REFRESH : displaying the current menu
      ➔ MENU_CHANGE or MENU_CONTINUE : nothing

➔ MENU_CONTINUE_COMMAND : retrieving the former command to the current one

Main process:

✔ if (there is a valid current command) :

➔ launching **Fct_Manage**()

✔ if (return code of **Fct_Manage**() is MENU_LEAVE) :
➔ current command and menu to 0 and leaving menus to the main screen.

✔ If (return code of Fct_Manage() is MENU_RESTORE_COMMAND) :
➔ retrieving former command to the current one

10.3 Functions

10.3.1 Internal functions

`fShutdown()`

Switch off the power supply.

`fSDCard()`

Display SDCard menu.

`RefreshItem(sel,isInverted)`

Display the text of the command.

`fyes()`

Records the answer 'Yes' for the current question.

`fno()`

Records the answer 'No' for the current question.

`MENU_Handler()`

`fQuit()`

Leaves current menu (stands for "cancel").

`fSetPllRange_Mgr()`

Set the level of the stable position calling MENU_SetLevel_Mgr().

`fSetBacklight_Mgr()`

Set the level of the backlight calling MENU_SetLevel_Mgr()

`fSetTime_Ini()`

Initialization function for time setting application.

`fSetTime_Mgr()`

Time setting application, returns MENU_CONTINUE or MENU_LEAVE.

fconfig()

Launch the general menu for configuration.

faboutIni()

Initialization function for the 'About' application.

fAboutMgr()

'Main' function for the 'About' application.

fInMems()

Select the MEMS as user input.

fInJoystick()

Select the JOYSTICK as user input.

fInBoth()

Select both the MEMS and the JOYSTICK as user input.

fInTchscr()

Select TOUCHSCREEN as input.

fCalibration()

Launch a touchscreen calibration

fTest_Mgr()

Hardware test application.

fSetInput()

Set if the commands are done by the joystick instead of the mems.

fSetSpeaker()

Set if the loudspeaker is active or not.

fSetBeep()

Set if the Beep is active or not.

fSetMenu_Ini()

Make the list of the fonts.

fSetMenu_Mgr

Set the Menu font (=DEFAULT font).

fInterface()

Launch the general menu for configuration.

fPower()

Launch the general menu for Power management.

fSetAutorun()

Set if the application autorun or not

DB_LED_Init()

Initialization of the GPIOs for the LEDs of the daughter card.

FctAppIni()

Show the list of loaded applications.

FctAppMgr()

Manage the selection of the application to be launch.

APP_LaunchAppli(index)

Launch an application.

APP_FindAppIndex(index)

Read the FAT in order to find the aplication corresponding to the menu index

10.3.2 APIs

MENU_Set(@tMenu)

Display provided menu.

MENU_Remove()

Remove current menu, clear screen and set pointer "on".

MENU_Question(@str,answer)

Dedicated menu for ask question and yes/no responses (answer: 1 for yes, 0 for no).

MENU_Print(str)

Display a popup menu with a string.

MENU_ClearCurrentCommand()

Set CurrentCommand to 0.

MENU_SetLevelTitle(@title)

Set the title of level menu managed by MENU_SetLevel_Mgr().

MENU_SetTextColor(color_t)

Set the colour used for text menu.

MENU_GetTextColor()

Return the colour used for text menu.

MENU_SetBGndColor(color_t)

Set the background color used for menu.

MENU_GetBGndColor()

Return the background color used for menu.

MENU_Quit()

Leave the current menu (stand for "cancel") and clear screen.

MENU_SetLevel_Ini()

Initialize a generic function to set a value in the range of [0,4]

MENU_ClearCurrentMenu()

Set CurrentMenu to 0

MENU_SetLevel_Mgr(@value,@value_range)

Generic function to set a value in the range of [0,4] (handling of the control)

MENU_SetAppliDivider(divider)

Set new value of the call time application divider. Upon each menu handler tick, CircleOS will have the opportunity to call your application or not, depending on this divider.
If divider = 1, your application will be called upon each menu handler occurrences, typically 10 ms.
If divider = 2 your application will be called only every two menu handler occurrences, typically 20 ms.

Note : if you want to call your application more frequently, you can reduce the menu handler divider, with the function :
UTIL_SetDividerHandler(MENU_SCHHDL_ID, 1);

Be careful with the menu handler divider value if you are managing menus in your application.

MENU_RestoreAppliDivider()
Restore the default divider (MENU) for the application divider.

10.4 Structures

tMenu()

List configuration information :

- .fdispTitle : presence of a title,
- .@ Title : title of the menu
- .NbItems : 0 to MENU_MAXITEMS,
- .LgMax : maximimum number of character (calculated by the manager),
- .XPos, .Ypos : position of the list (updated by the manager),
- .XSize, .Ysize : size of the list (updated by the manager)
- .SelectedItem : current selected item, (updated by the manager),
- .tMenuItem : array of MENU_MAXITEMS Menu_Items

tmenuItem()

- .Text : @ text of the item
- .*Fct_Init : @ init function
- .*Fct_Manage : @ manager function
- .fMenuFlag : flags menu

- `MENU_MAXITEM` = 7.

11. File system

11.1 Introduction

The CircleOS provides some functions in the CircleOS API that make generic the simple accesses to SD card files.

11.2 CircleOS < 4.4

The file system is based on DOSFS, that is a Free FAT12/FAT16/FAT32 Filesystem developed by Edwards Lewin.

The implementation includes 2 files:

- dosfs.c : file system,
- fs.c : OS layer of the file system.

11.2.1 DOSFS presentation

DOSFS Embedded FAT-Compatible Filesystem

(C) 2005 Lewin A.R.W. Edwards (sysadm@zws.com)

Features:

- Supports FAT12, FAT16 and FAT32 volumes
- Supports storage devices up to 2048Gbytes in size (LBA32)
- Supports devices with or without MBRs (hard disks vs. floppy disks or ZIP drives formatted as "big floppies")
- Supports multiple partitions on disks with MBRs
- Supports subdirectories
- Can be operated with a single global 512-byte sector buffer
- Fully reentrant code (assuming the underlying physical device driver is reentrant and global sector buffers are not used). There are no global variables in the filesystem
- Does not perform any memory allocation
- Partial support for random-access files

Notes

Physical 32-bit sector numbers are used throughout. Therefore, the CHS geometry (if any) of the storage media is not known to DOSFS. Your sector r/w functions may need to query the CHS geometry and perform mapping.

File timestamps set by DOSFS are always 1:01:00am on Jan 1, 2006.

FILEINFO structures contain a pointer to the corresponding VOLINFO used to open the file, mainly in order to avoid mixups but also to obviate the need for an extra parameter to every file read/write.

DOSFS assumes that the VOLINFO won't move around. If you need to move or destroy VOLINFOs pertaining to open files, you'll have to fix up the pointer in the FILEINFO structure yourself.

The subdirectory delimiter is a forward slash ('/') by default. The reason for this is to avoid the common programming error of forgetting that backslash is an escape character in C strings; i.e. "\MYDIR\FILE" is NOT what you want; "\\MYDIR\\FILE" is what you wanted to type. If you are porting DOS code into an embedded environment, feel free to change this #define.

DOSFS does not have a concept of "current directory". A current directory is owned by a process, and a process is an operating system concept.

DOSFS is a filesystem library, not an operating system. Therefore, any path you provide to a DOSFS call is assumed to be relative to the root of the volume.

There is no call to close a file or directory that is open for reading or writing. You can simply destroy or reuse the data structures allocated for that operation; there is no internal state in DOSFS so no cleanup is necessary. Similarly, there is no call to close a file that is open for writing. (Observe that dosfs.c has no global variables. All state information is stored in data structures provided by the caller).

Long file name are not supported, bit only DOS format name (8.3).

11.2.2 Structures

```
typedef enum STORAGE_device
{
   MMCSD_SDIO,
   INTERNAL_FLASH // currently not used
} eSTORAGE_device;

/**
* @struct   _tagVOLINFO
* @brief    Volume information structure.
**/
typedef struct _tagVOLINFO
{
   u8 unit;             // unit on which this volume resides
   u8 filesystem;       // formatted filesystem

   /* These two fields aren't very useful, so support for them has been commented out to*/
   /* save memory. (Note that the "system" tag is not actually used by DOS to determine*/
   /* filesystem type - that decision is made entirely on the basis of how many clusters*/
   /* the drive contains. DOSFS works the same way).*/
   /* See tag: OEMID in dosfs.c*/
   /*  u8 oemid[9];       // OEM ID ASCIIZ*/
   /*  u8 system[9];      // system ID ASCIIZ*/
   u8  label[12];         /* volume label ASCIIZ*/
   u32 startsector;       /* starting sector of filesystem*/
   u8  secperclus;        /* sectors per cluster*/
   u16 reservedsecs;      /* reserved sectors*/
   u32 numsecs;           /* number of sectors in volume*/
   u32 secperfat;         /* sectors per FAT*/
   u16 rootentries;       /* number of root dir entries*/

   u32 numclusters;       /* number of clusters on drive*/

   /* The fields below are PHYSICAL SECTOR NUMBERS.*/
   u32 fat1;              /* starting sector# of FAT copy 1*/
   u32 rootdir;           /* starting sector# of root directory (FAT12/FAT16) or cluster (FAT32)*/
   u32 dataarea;          /* starting sector# of data area (cluster #2)*/
} VOLINFO, *PVOLINFO;
```

```
/**
* @struct   _tagDIRENT
* @brief    Directory entry structure.
*
* Note: if name[0] == 0xe5, this is a free dir entry
*       if name[0] == 0x00, this is a free entry and all subsequent entries are free
*       if name[0] == 0x05, the first character of the name is 0xe5 [a kanji nicety]
*
*   Date format: bit 0-4  = day of month (1-31)
*              bit 5-8  = month, 1=Jan..12=Dec
*              bit 9-15 = count of years since 1980 (0-127)
*   Time format: bit 0-4  = 2-second count, (0-29)
*              bit 5-10 = minutes (0-59)
*              bit 11-15= hours (0-23)
**/

typedef struct _tagDIRENT
{
   u8 name[11];          /* filename*/
   u8 attr;              /* attributes (see ATTR_* constant definitions)*/
   u8 reserved;          /* reserved, must be 0*/
   u8 crttimetenth;      /* create time, 10ths of a second (0-199 are valid)*/
   u8 crttime_l;         /* creation time low byte*/
   u8 crttime_h;         /* creation time high byte*/
   u8 crtdate_l;         /* creation date low byte*/
   u8 crtdate_h;         /* creation date high byte*/
   u8 lstaccdate_l;      /* last access date low byte*/
   u8 lstaccdate_h;      /* last access date high byte*/
   u8 startclus_h_l;     /* high word of first cluster, low byte (FAT32)*/
   u8 startclus_h_h;     /* high word of first cluster, high byte (FAT32)*/
   u8 wrttime_l;         /* last write time low byte*/
   u8 wrttime_h;         /* last write time high byte*/
   u8 wrtdate_l;         /* last write date low byte*/
   u8 wrtdate_h;         /* last write date high byte*/
   u8 startclus_l_l;     /* low word of first cluster, low byte*/
   u8 startclus_l_h;     /* low word of first cluster, high byte*/
   u8 filesize_0;        /* file size, low byte*/
   u8 filesize_1;        /**/
   u8 filesize_2;        /**/
   u8 filesize_3;        /* file size, high byte*/
} DIRENT, *PDIRENT;


/**
* @struct   _tagDIRINFO
* @brief    Directory search structure.
```

```
**/
typedef struct _tagDIRINFO
{
   u32 currentcluster;          /* current cluster in dir*/
   u8  currentsector;           /* current sector in cluster*/
   u8  currententry;            /* current dir entry in sector*/
   u8* scratch;                 /* ptr to user-supplied scratch buffer (one sector)*/
   u8  flags;                   /* internal DOSFS flags*/
} DIRINFO, *PDIRINFO;

/**
* @struct   _tagFILEINFO
* @brief    File handle structure.
**/
typedef struct _tagFILEINFO
{
   PVOLINFO volinfo;            /* VOLINFO used to open this file*/
   u32 dirsector;               /* physical sector containing dir entry of this file*/
   u8  diroffset;               /* # of this entry within the dir sector*/
   u8  mode;                    /* mode in which this file was opened*/
   u32 firstcluster;            /* first cluster of file*/
   u32 filelen;                 /* byte length of file*/
   u32 cluster;                 /* current cluster*/
   u32 pointer;                 /* current (BYTE) pointer*/
} FILEINFO, *PFILEINFO;
```

11.2.3 APIs

`FS_Mount((enum STORAGE_device device )`

Initializes and connects selected device to file system (only SDCard supported)

`FS_Unmount(enum STORAGE_device device )`

Unmount the SD card device (Not yet implemented).

`FS_OpenFile(PVOLINFO volinfo, u8 *path, u8 mode, PFILEINFO fileinfo )`

Open a file

`FS_ReadFile(PFILEINFO fileinfo, u8 *buffer, u32 *successcount, u32 len)`

Read from the file

`FS_WriteFile (PFILEINFO fileinfo, u8 *buffer, u32 *successcount, u32 len )`

Write to the file

`FS_Close(PFILEINFO fileinfo)`

Close the file (Not yet implemented).

`FS_Delete(PVOLINFO volinfo, u8 *path)`

Delete the file

FS_Seek(PFILEINFO fileinfo, u32 offset)

Move into the file

FS_GetNextEntry(PVOLINFO volinfo, PDIRINFO dirinfo, PDIRENT dirent)

Search next entry of the directory.

FS_OpenDirectory(PVOLINFO volinfo, u8 *dirname, PDIRINFO dirinfo)

Open a directory

FS_GetVolumeInfo(u8 unit, u32 startsector, PVOLINFO volinfo)

Gets the volume informations for filling volinfo structure.

FS_Explorer_Ini()

Checks the presence of the SDCARD, initializes various structures and lists the root directory.

FS_Explorer()

Navigation into the SDCARD folders, through a list, and selection of a file.

FS_GetSDCardCurrentPath()

Get the Currentpath of the SDCard, updated during navigation with the explorer.

FS_GetSDCardVolInfo()

Get the volume informations of the SDCard, structure populated by the

FS_Explorer_Ini()

This structure is necessary for file access, (FS_OpenFile, FS_ReadFile, FS_Seek...).

FS_GetPathFilter()

Get the filter applied to the file type during exploring the SDCard

FS_SetPathFilter(u8 *filter)

Set the filter applied to the file type during exploring the SDCard. A null pointer indicates no filter

11.3 CircleOS >= 4.4

The file system has been changed to FatFs.

The implementation includes 4 files:

- ff.c : file system,
- fs.c : OS layer of the file system,
- diskio.c : low level disk I/O module for SDCard access,
- ccsbcs.c : Unicode - Local code bidirectional converter.

11.3.1 FatFs presentation

FatFs is a generic FAT file system module for small embedded systems, developped by ChaN.

The FatFs is written in compliance with ANSI C and completely separated from the disk I/O layer. Therefore it is independent of hardware architecture. It can be incorporated into low cost microcontrollers, such as AVR, 8051, PIC, ARM, Z80, 68k and etc..., without any change.

Features

- Windows compatible FAT file system.
- Platform independent. Easy to port.
- Very small footprint for code and work area.
- Various configuration options:
    - Multiple volumes (physical drives and partitions).
    - Multiple ANSI/OEM code pages including DBCS.
    - Long file name support in ANSI/OEM or Unicode.
    - RTOS support.
    - Multiple sector size support.
    - Read-only, minimized API, I/O buffer and etc...

11.3.2 Configuration

The version used is R0.08a.

The FatFS is fully configurable for optimization by modifying the “ffconf.h” file.

Here is the configuration implemented Into the CircleOS by can be modified.

```
#define _FS_TINY            0       /* 0:Normal or 1:Tiny */
#define _FS_READONLY        0       /* 0:Read/Write or 1:Read only */
#define _FS_MINIMIZE        0       /* Full functions */
#define _USE_STRFUNC        2       /* 0:Disable or 1/2:Enable */
#define _USE_MKFS   0               /* 0:Disable or 1:Enable */
#define _USE_FORWARD        0       /* 0:Disable or 1:Enable */
#define _USE_FASTSEEK       0       /* 0:Disable or 1:Enable */

#define _CODE_PAGE 437              /* U.S. (OEM) */

#define _USE_LFN    1               /* 1 : Enable LFN with static working buffer on the BSS. Always
NOT reentrant. */
#define _MAX_LFN    255             /* Maximum LFN length to handle (12 to 255) */
#define _LFN_UNICODE        0       /* 0:ANSI/OEM or 1:Unicode */

#define _FS_RPATH   0               /* 0: Disable relative path feature and remove related
functions. */

#define _VOLUMES    1               /* Number of volumes (logical drives) to be used. */
#define _MAX_SS     512             /* Maximum sector size to be handled.
#define _MULTI_PARTITION    0       /* 0:Single partition or 1:Multiple partition */
#define _USE_ERASE  0               /* To enable sector erase feature, set _USE_ERASE to 1. */

#define _WORD_ACCESS        0       /* 0: Byte-by-byte access. */

#define _FS_REENTRANT       0       /* 0:Disable or 1:Enable */

#define _FS_SHARE   2               /* The value defines how many files can be opened
simultaneously. */
```

11.3.3 Stuctures

All the previous define structures and functions are compatible with the new file system. The fs.c file converts and fills all necessary structures in order to stay compatible.

11.3.4 New APIs

Some new functions has been added and listed below.

```
u32 FS_Tell( PFILEINFO fileinfo )
```

Retrieve file pointer on the current position. (to use with FS_Seek later).

```
u32 FS_Gets(char* ptr, int len, PFILEINFO fileinfo)
```

Get a string from the file.

```
u32 FS_Size( PFILEINFO fileinfo )
```

Retrieve file size.

```
u32 FS_Eof( PFILEINFO fileinfo )
```

Check the end of the file.

```
u32 FS_FileCopy( char* dest, char* src )
```

Makes a file copy, by 512 bytes packets long.

```
u32 FS_FileCmp( char* fn1, char* fn2 )
```

Compares two files, by 512 bytes packets long.

11.4 Example

Take a look at the project example "**FileSystemTest**" provided on the STM32 Circle web site, that implements all the principal functions on all Primer platforms which have a SD Card interface.

http://www.stm32circle.com/

12. Utilities

12.1 Files

Util.c
util_spe.c : shared functions between STM32 platforms
util_spe2.c : platform specific functions

This modules provides common and generic useful functions.

12.2 Various APIs

`const u8* UTIL_GetVersion (void)`

Get CircleOS version in string format.

`u16 UTIL_GetPrimerType (void)`

Get the current type of the PRIMER. (1 = PRIMER 1, 2 = PRIMER 2, 4 = OPEN4.)

`bool UTIL_IsStandAloneMode (void)`

Get the current mode of the Primer. Useful for low power platforms that can run in standalone mode, outside the Primer base (STM8L, STM32L...)

`UTIL_SetPll (enum eSpeed speed)`

Set clock frequency (lower to save energy)

`eSpeed UTIL_GetPll (void)`

Get CPU clock frequency

`UTIL_SetSchHandler (enum eSchHandler Ix, tHandler pHDL)`

Get the current SCHEDULER handler. With UTIL_SetSchHandler(), these functions allow to take the control of the different handler. You can:

- replace them (get-Set)by your own handler
- disable a handler: UTIL_SetSchHandler(Ix,0);
- create a new handler (using the unused handlers). See scheduler.c to understand further...

`tHandler UTIL_GetSchHandler (enum eSchHandler Ix)`

Get the current SCHEDULER handler. With UTIL_SetSchHandler(), these functions allow to take the control of the different handler. You can:

- replace them (get-Set)by your own handler
- disable a handler: UTIL_SetSchHandler(Ix,0);
- create a new handler (using the unused handlers). See scheduler.c to understand further...

`UTIL_SetDividerHandler(enum eSchHandler IDx, u16 divider)`

Set the current divider for a SCHEDULER handler.

Note : if you want to call your application more frequently, you can reduce the menu handler divider, with the function

Be careful with the menu handler divider value if you are managing menus in your application.

`u16 UTIL_GetDividerHandler(enum eSchHandler Idx)`

Get the current divider for a SCHEDULER handler.

`UTIL_SetIrqHandler (s32 Offs, tHandler pHDL)`

Get the current IRQ handler. Since (V1.6) the vector table is relocated in RAM, the vectors can be easily modified by the applications.

`tHandler UTIL_GetIrqHandler (s32 Offs)`

Redirect an IRQ handler.

`uint_t UTIL_GetAppAddress (const u8 *AppName)`

Get the address of an application in the FAT.

`u16 UTIL_GetBat (void)`

Return the battery tension in mV.

`UTIL_GetBatStatus()`

Reads the status of the battery charger :

0 : battery is absent or dead (green on and red on)
1 : charged (green on and red off)
2 : battery is being charged (green off and red on)
3 : power supplied by battery (green off and red off) = normal mode

`UTIL_SetTempMode (bool mode)`

Set the temperature mode (F/C), for platforms that provide CPU temperature on the main screen.

`u16 UTIL_GetTemp (void)`

Return the Temperature: degrees / 10, Celcius or Fahrenheit.

`backup_t UTIL_ReadBackupRegister (index_t BKP_DR)`

Reads data from the specified Data Backup Register. (cf § Erreur : source de la référence non trouvée Erreur : source de la référence non trouvéeErreur : source de la référence non trouvée)

`UTIL_WriteBackupRegister (index_t BKP_DR, backup_t Data)`

Writes data to the specified Data Backup Register. (cf § Erreur : source de la référence non trouvée Erreur : source de la référence non trouvéeErreur : source de la référence non trouvée)

`UTIL_uint2str (u8 *ptr, uint_t X, len_t digit, bool fillwithzero)`

Convert an unsigned integer into a string. Using this function leads to much smaller code than using the sprintf() function from the C library.

`UTIL_int2str (u8 *ptr, int_t X, len_t digit, bool fillwithzero)`

Convert a signed integer into a string. Using this function leads to much smaller code than using the sprintf() function from the C library.

12.3 Save screen

An utility allows to capture/save the current screen on the SD card

`UTIL_SaveScreenBMP()`

The format is the standard Windows 24-bit BMP (no compression).

IMPORTANT NOTE : The BMP format requires 32 bit alignment between lines.
This alignment is made automatically as long as width is a multiple of 4. If NOT, padding with 0 should be added to the functions (not done in this first version).

Take a look at the project example "**ScreenShot**" provided on the STM32 Circle web site, that implements "*UTIL_SaveScreenBMP*" on all Primer platforms which have a SD Card interface.

12.4 High priority timer interrupt handling

12.4.1 Principle

The following function has been added :

UTIL_SetTimer (long int *millisec*, void (**fTimerHandler*) (void));

where:

- *millisec* specifies the delay (in millisecond) to wait for before calling *fTimerHandler*
- *fTimerHandler* is the function to be called as soon as the *millisec* delay is past.

Note:

- The function is called only once. For a periodic call, you have to call again **UTIL_SetTimer** from the handler (*fTimerHandler)* itself.
- The delay is measured by an interrupt service of a timer that has a higher priority than the SYSTICK interrupt priority. Therefore, this function can be used to manage a timeout in any scheduler function (e.g. Inside *Application_Handler*()...).
- The millisecond is NOT an exact multiple of the used timer interrupt. But you can assume that the function handler will be called between *millisec* and (*millisec*+1) ms.
- Only one event can be managed at the same time. Any call to **UTIL_SetTimer** will overwrite the current expected event.
- When exiting an application, it is advised to call **UTIL_SetTimer** (*0*, *0*); to delete any expected event.

12.4.2 Example 1: Timeout

```
int flagTimeout = 0;
void MyHandler ( void ) {        flagTimeout = 1;        }
```

```
void Application_Handler ( void )
	{
	#define TIMEOUT 10
	/*.... */
	flagTimeout = 0;
	UTIL_SetTimer ( TIMEOUT,  MyHandler  );
	while (!flagTimeout)
		{
		/* wait for a reception or whatever... */
		if (flagReceived)
			{
			break;
			}
		}
	}
```

12.4.3 Example 2: Periodic call

```
void MyHandler ( void )
	{
	#define PERIOD 10
	/* ... do something ... */
	UTIL_SetTimer ( PERIOD,  MyHandler  );
	}
void Application_Init ( void )
	{
	UTIL_SetTimer ( PERIOD,  MyHandler  ); //start the periodic function
	}
```

13. Cx extension

13.1 Introduction

OP4 and Primer2 feature the same extension connector. Most of the pins have been specified as general purpose IOs, or are dedicated for a specific hardware interface : USART, SPI, I²C,...

The CircleOS provides some functions in the CircleOS API that make generic the simple accesses to IO pins of the extension board (and to the generic communication interfaces).

13.2 Platform pin out

13.2.1 Primer2

Pin	Printed name (*)	Pin	Name	Function	Remark
1	V2V8	Vcc	VCC2V8		Voltage provides by Primer
2	GND	GND	GND		
3	SCL	PB.6	CX_I2CSCL	I2C1_SCL / TIM4_CH1 / USART1_TX	
4	SDA	PB.7	CX_I2CSDA	I2C1_SDA / TIM4_CH2 / USART1_RX	
5	MISO	PB.14	AUDIO_SPI_MISO	SPI2_MISO / TIM1_CH2N / USART3_RTS	SPI2 / I2S2 is shared with audio codec.
6	SD	PB.15	AUDIO_I2S2_SD	SPI2_MOSI / I2S2_SD / TIM1_CH3N	
7	SCK	PB.13	AUDIO_I2S2_SCK	SPI2_SCK / I2S2_CK / USART3_CTS	
8	WS	PB.12	AUDIO_I2S2_WS	SPI2_NSS / I2S2_WS / USART3_CK	
9	CANH		CX_CANH		CAN transceiver outputs (not soldered) are located on the connector pins, and use the micro-controller pins : - Rx: PD0, - Tx: PD1
10	CANL		CX_CANL		
11	ADC1	PC.4	CX_ADC1	ADC12_IN14	
12	ADC2	PC.5	CX_ADC2	ADC12_IN15	
13	A_TIM	PB.0	CX_ADC_TIM	ADC12_IN8 / TIM3_CH3	
14	CTS	PA.0	CX_USART_CTS	USART2_CTS / ADC123_IN0 / TIM5_CH1	
15	RTS	PA.1	CX_USART_RTS	USART2_RTS / ADC123_IN1 / TIM5_CH2 / TIM2_CH2	
16	TX	PA.2	CX_USART_TX	USART2_TX / TIM5_CH3 / ADC123_IN2 / TIM2_CH3	
17	CK	PA.4	CX_USART_CK	USART2_CK/ DAC_OUT1 / ADC12_IN4	
18	RX	PA.3	CX_USART_RX	USART2_RX / TIM5_CH4 / ADC123_IN3 / TIM2_CH4	
19	VEXT		VCC_EXT		Primer2 took an external power supply
20	GND	VSS	GND		Ground

(*) : name printed on the Extension Card

13.2.2 Open4 STM32E / STM32G

Pin	Printed name (*)	Pin	Name	Function	Remark
1	V2V8	Vcc	VCC		EvoPrimer provides 3V1 instead of 2V8
2	GND	GND	GND		
3	SCL	PB.6	CX_I2CSCL	I2C1_SCL / TIM4_CH1 / USART1_TX	
4	SDA	PB.7	CX_I2CSDA	I2C1_SDA / TIM4_CH2 / USART1_RX	
5	MISO	PB.4	SPI3_MISO	SPI3_MISO / TIM3_CH1 / SPI1_MISO	SPI3 is shared with SWV trace debug output Cx API uses DMA2 channel 1 for Rx, Cx API uses DMA2 channel 2 for Tx.
6	SD	PB.5	SPI3_MOSI / I2S3_SD	SPI3_MOSI / I2S3_SD / TIM3_CH2 / SPI1_MOSI	
7	SCK	PB.3	SWO	SPI3_SCK / I2S3_CK / TRACESWO / TIM2_CH2 / SPI1_SCK	
8	WS	PA.15	JTDI_I2S3_WS	SPI3_NSS / I2S3_WS / SPI1_NSS	
9	CANH		CX_CANH	CAN	CAN transceiver outputs (not soldered) are located on the connector pins, and use the micro-controller pins : - Rx: PB8, - Tx: PB9
10	CANL		CX_CANL	CAN	
11	ADC1	PC.4	CX_ADC1	ADC12_IN14	
12	ADC2	PC.5	CX_ADC2	ADC12_IN15	
13	A_TIM	PB.0	CX_ADC_TIM	ADC12_IN8 / TIM3_CH3	
14	CTS	PA.0	CX_USART_CTS	USART2_CTS / ADC123_IN0 / TIM5_CH1	
15	RTS	PA.1	CX_USART_RTS	USART2_RTS / ADC123_IN1 / TIM5_CH2 / TIM2_CH2	
16	TX	PA.2	CX_USART_TX	USART2_TX / TIM5_CH3 / ADC123_IN2 / TIM2_CH3	Cx API uses DMA1 channel 7 for Tx
17	CK	N/A	P_BUTTON		Push Button may allow Extension Board to wake up the Base.
18	RX	PA.3	CX_USART_RX	USART2_RX / TIM5_CH4 / ADC123_IN3 / TIM2_CH4	Cx API uses DMA1 channel 6 for Rx
19	VEXT	N/A	VBAT		EvoPrimer provides the base battery voltage.
20	GND	VSS	GND		Ground

(*) : name printed on the Extension Card

13.2.3 Open4 STM32C

Pin	Printed name (*)	Pin	Name	Function	Remark
1	V2V8	Vcc	VCC		EvoPrimer provides 3V1 instead of 2V8
2	GND	GND	GND		
3	SCL	PB.8	CX_I2CSCL	TIM4_CH3 / I2C1_SCL / CAN1_RX	I2C1 need s remapping
4	SDA	PB.9	CX_I2CSDA	TIM4_CH4 I2C1_SDA / CAN1_TX	
5	MISO	PC.11	SPI3_MISO	UART4_RX / USART3_RX / SPI3_MISO	SPI3 is shared with SD Card SPI3 need s remapping Cx API uses DMA2 channel 1 for Rx, Cx API uses DMA2 channel 2 for Tx.
6	SD	PC.12	SPI3_MOSI	UART5_TX / USART3_CK / SPI3_MOSI / I2S3_SD	
7	SCK	PC.10	SPI3_SCK	UART4_TX / USART3_TX / SPI3_SCK / I2S3_CK	
8	WS	PA.15	JTDI_I2S3_WS	SPI3_NSS / I2S3_WS / SPI1_NSS	
9	CANH		CX_CANH		First CAN transceiver outputs are located on the connector pins, and use the micro-controller pins : - Rx: PD0, - Tx: PD1
10	CANL		CX_CANL		
11	ADC1	PC.4	CX_ADC1	ADC12_IN14	
12	ADC2	PC.5	CX_ADC2	ADC12_IN15	
13	A_TIM	PB.0	CX_ADC_TIM	ADC12_IN8 / TIM3_CH3	
14	CTS		CX_USART_CTS		Second CAN transceiver outputs are located on the connector pins, and use the micro-controller pins : - Rx: PB5, - Tx: PB6
15	RTS		CX_USART_RTS		
16	TX	PA.2	CX_USART_TX	USART2_TX / TIM5_CH3 / ADC12_IN2 / TIM2_CH3	Cx API uses DMA1 channel 7 for Tx
17	CK	N/A	P_BUTTON		Push Button may allow Extension Board to wake up the Base.
18	RX	PA.3	CX_USART_RX	USART2_RX / TIM5_CH4 / ADC12_IN3 / TIM2_CH4	Cx API uses DMA1 channel 6 for Rx
19	VEXT	N/A	VBAT		EvoPrimer provides the base battery voltage.
20	GND	VSS	GND		Ground

(*) : name printed on the Extension Card

13.2.4 Open4 STM32L

Pin	Printed name (*)	Pin	Name	Function	Remark
1	V2V8	Vcc	VCC		EvoPrimer provides 3V1 instead of 2V8
2	GND	GND	GND		
3	SCL	PB.6	CX_I2CSCL	I2C1_SCL / TIM4_CH1 / USART1_TX	
4	SDA	PB.7	CX_I2CSDA	I2C1_SDA / TIM4_CH2 / USART1_RX	
5	MISO	PB.14	SDCARD_MISO	SPI2_MISO / USART3_RTS / ADC_IN20 / TIM9_CH2	SPI2 is shared with SD Card Cx API uses DMA1 Channel 4 for Rx, Cx API uses DMA1 Channel 5 for Tx. SPI 2 NSS not available
6	SD	PB.15	SDCARD_MOSI	SPI2_MOSI / TIM1_CH3N / ADC_IN21 / TIM11_CH1	
7	SCK	PB.13	SDCARD_SCK	SPI2_SCK / USART3_CTS / ADC_IN19 / TIM9_CH1	
8	WS	PB.8	CX_TIM1	TIM4_CH3 / TIM10_CH1	
9	CANH	PD.0	CX_TIM2	TIM9_CH1	
10	CANL	PA.4	CX_ADC_DAC	ADC_IN4 / DAC_OUT1	
11	ADC1	PA.2	CX_ADC1	ADC_IN2	ADC already used by CircleOS
12	ADC2	PA.3	CX_ADC2	ADC_IN3	
13	A_TIM	PA.1	CX_ADC_TIM	ADC_IN1/ TIM2_CH2/	
14	CTS	PD.3	CX_USART_CTS	USART2	
15	RTS	PD.4	CX_USART_RTS	USART2	
16	TX	PD.5	CX_USART_TX	USART2	Cx API uses DMA1 Channel 7
17	CK	N/A	P_BUTTON	N/A	Push Button may allow Extension Board to wake up the Base.
18	RX	PD.6	CX_USART_RX	USART2	Cx API uses DMA1 Channel 6
19	VEXT	N/A	VBAT		EvoPrimer provides the base battery voltage.
20	GND	VSS	GND		Ground

(*) : name printed on the Extension Card

13.2.5 Open4 STM3240G

Pin	Printed name (*)	Pin	Name	Function	Remark
1	V2V8	Vcc	VCC		EvoPrimer provides 3V1 instead of 2V8
2	GND	GND	GND		
3	SCL	PH.4	CX_I2CSCL	I2C2_SCL	
4	SDA	PH.5	CX_I2CSDA	I2C2_SDA	
5	MISO	PB.4	CX_SPI1_MISO	SPI1_MISO / TIM3_CH1	SPI1 is shared with SWV trace debug output Cx API uses DMA2 Channel 3 Stream 0 for Rx, Cx API uses DMA2 Channel 3 Stream 5 for Tx.
6	SD	PA.7	CX_SPI1_MOSI	SPI1_MOSI / TIM14_CH1 / TIM3_CH2 / ADC12_IN7	
7	SCK	PB.3	CX_SPI1CK_SWO	SPI1_SCK / TIM2_CH2	
8	WS	PA.15	CX_SPI1_NSS	SPI1_NSS	
9	CANH	PA.11	CAN_RX	CAN1_RX / TIM1_CH4	There is no CAN transceiver on the card
10	CANL	PA.12	CAN_TX	CAN1_TX	
11	ADC1	PA.4	CX_ADC1	ADC12_IN4 / DAC1_OUT	
12	ADC2	PA.2	CX_ADC2	ADC123_IN2 / TIM5_CH3 / TIM9_CH1 / TIM2_CH3	
13	A_TIM	PB.8	CX_TIM	TIM4_CH3 / TIM10_CH1	
14	CTS	PG.13	CX_USART_CTS	USART6_CTS	
15	RTS	PG.12	CX_USART_RTS	USART6_RTS	
16	TX	PG.14	CX_USART_TX	USART6_TX	Cx API uses DMA2 Channel 5 Stream 6
17	CK	N/A	P_BUTTON		Push Button may allow Extension Board to wake up the Base.
18	RX	PC.7	CX_USART_RX	USART6_RX / TIM8_CH2 / TIM3_CH2	Cx API uses DMA2 Channel 5 Stream 2
19	VEXT	N/A	VBAT		EvoPrimer provides the base battery voltage.
20	GND	VSS	GND		Ground

(*) : name printed on the Extension Card

13.2.6 Open4 STM32429x

Pin	Printed name (*)	Pin	Name	Function	Remark
1	V2V8	Vcc	VCC		EvoPrimer provides 3V1 instead of 2V8
2	GND	GND	GND		
3	SCL	PB.8	CX_I2CSCL	TIM4_CH3 / TIM10_CH1 / I2C1_SCL / CAN1_RX	
4	SDA	PB.9	CX_I2CSDA	TIM4_CH4 / TIM11_CH1 / I2C1_SDA / CAN1_TX	
5	MISO	PA.6	CX_SPI_MISO	SPI1_MISO / TIM3_CH1 / TIM13_CH1 / ADC12_IN6	Cx API uses DMA2 Channel 3 Stream 0 for Rx, Cx API uses DMA2 Channel 3 Stream 5 for Tx.
6	SD	PA.7	CX_SPI_MOSI	SPI1_MOSI / TIM1_CH1N / TIM3_CH2 / TIM8_CH1N / TIM14_CH1 / ADC12_IN7	
7	SCK	PA.5	CX_SPI1CK	SPI1_SCK / TIM2_CH1 / TIM8_CH1N / ADC12_IN5 / DAC_OUT2	
8	WS	PA.4	CX_SPI1_NSS	SPI1_NSS / TIM2_CH1	
9	CANH	PA.11	CAN_RX	CAN1_RX / OTG_FS_DM	There is no CAN transceiver on the card
10	CANL	PA.12	CAN_TX	CAN1_TX / OTG_FS_DP	
11	ADC1	PA.3	CX_ADC1	ADC123_IN3 / TIM2_CH4 / TIM5_CH4 / TIM9_CH2 / USART2_RX	
12	ADC2	PA.2	CX_ADC2	ADC123_IN2 / TIM2_CH3 / TIM5_CH3 / TIM9_CH1, USART2_TX	
13	A_TIM	PB.7	CX_TIM	TIM4_CH3 / TIM10_CH1	
14	CTS	PG.13	CX_USART_CTS	USART6_CTS	
15	RTS	PG.12	CX_USART_RTS	USART6_RTS	
16	TX	PG.14	CX_USART_TX	USART6_TX	Cx API uses DMA2 Channel 5 Stream 6
17	CK	N/A	P_BUTTON		Push Button may allow Extension Board to wake up the Base.
18	RX	PC.7	CX_USART_RX	USART6_RX / TIM3_CH2, / TIM8_CH2	Cx API uses DMA2 Channel 5 Stream 2
19	VEXT	N/A	VBAT		EvoPrimer provides the base battery voltage.
20	GND	VSS	GND		Ground

(*) : name printed on the Extension Card

13.3 Files

Cx.c : generic API functions,
Cx_spec.c : spécific functions by µC family (STM32),
Cx_spec_low_level.c : hardware platform specific functions.

13.4 Principles

The CircleOS functions are supposed NOT to be device dependent (i.e. they are supposed to work with any OP4 daughter board (and with a Primer2 as well).

There are only 3 generic functions :

- **`int32 CX_Configure( tCX_ID what, int32 param1, int32 param2 )`**
- **`int32 CX_Read( tCX_ID what, int32 param1, int32 param2 )`**
- **`int32 CX_Write( tCX_ID what, int32 param1, int32 param2 )`**

All these 3 functions return 0 when the operation succeeded. If it fails, the functions will return a non-zero value.

The first parameter *what* specifies the interface pin(s) to address (for GPIO's, the index is the pin number on the extension connector). The two others are generally pointers and they depend on the type of "*what*".

what belongs to

```
enum {
        CX_GPIO_PIN3 = 3,
        CX_GPIO_PIN4 = 4,
        ...
        CX_GPIO_PIN18 = 18,
        CX_USART,
        CX_SPI,
        CX_I2C,
        CX_ADC1,
        CX_ADC2
        }
tCX_ID;
```

Note : CX I2C and CX_CAN are not yet managed by CircleOS.

13.5 Using GPIO's

Depending on the manufacturer, on the device, GPIOs could have many different configurations (for speed, impedance, ...). CircleOS allows only basic settings. If the programmer needs any special configuration for the device peripherals, he will still have to configure them in his application.

Note that when a pin is used for an alternate function (USART, ...), there is no need to configure it as a GPIO. It will be done by CircleOS when configuring this alternate function.

13.5.1 Configuration

int32 CX_Configure (tCX_ID *what*, int32 *param1*, int32 *param2*)

what is one of CX_GPIO_PINx

param1 is an integer matching with the following enum type:

```
typedef enum
{
        CX_GPIO_Mode_IN_HIZ = 0,
        CX_GPIO_Mode_IN_PD = 1,
        CX_GPIO_Mode_IN_PU = 2,
        CX_GPIO_Mode_OUT_OD = 3,
        CX_GPIO_Mode_OUT_PP = 4
}
tCX_GPIO_Mode;
```

param2 is not used. Supposed to be always 0 (reserved for future use).

Example:

```
CX_Configure( CX_GPIO_PIN4, CX_GPIO_Mode_IN_HIZ, 0 );
```

13.5.2 Reading a pin

int32 CX_Read (CX_ID *what*, int32 *param1*, int32 *param2*)

what is one of CX_GPIO_PINx configured as an input (CX_GPIO_Mode_IN_xx)

Note : that CX_Read can be used on INPUT and OUTPUT configured pins.

param1 is the address of the return status, either CX_GPIO_HI or CX_GPIO_LOW with

```
#define CX_GPIO_LOW  (0)
#define CX_GPIO_HI  (1)
```

param2 is not used. Supposed to be always 0 (reserved for future use).

Example:

```
int32 state;

CX_Configure(  CX_GPIO_PIN5,  CX_GPIO_Mode_IN_HIZ, 0);
CX_Read( CX_GPIO_PIN5, &state, 0 );

if ( state ==  CX_GPIO_HI )...
```

13.5.3 Writing a pin

int32 CX_Write (CX_ID *what*, int32 *param1*, int32 *param2*)

what is one of CX_GPIO_PINx configured as an input (CX_GPIO_Mode_IN_xx)

param1 is either CX_GPIO_HI or CX_GPIO_LOW with

```
#define CX_GPIO_LOW  (0)
#define CX_GPIO_HIGH (1)
```

param2 is not used. Supposed to be always 0 (reserved for future use).

```
Example:

      CX_Configure ( CX_GPIO_PIN4,  CX_GPIO_Mode_OUT_PP, 0);
      CX_Write ( CX_GPIO_PIN4, CX_GPIO_LOW, 0);
```

13.6 Using the ADC

CircleOS allows only basic settings (no DMA acquisition). If the programmer needs any special configuration for the device peripherals, he will still have to configure them in his application.

13.6.1 Configuration

int32 CX_Configure (tCX_ID *what*, int32 *param1*, int32 *param2*)

what is CX_ADC1 or CX_ADC2

param1 and param2 are not used. Supposed to be always 0 (reserved for future use).

Example:

```
CX_Configure( CX_ADC1, 0, 0 );
```

13.6.2 Reading a value

int32 CX_Read (CX_ID *what*, int32 *param1*, int32 *param2*)

what is CX_ADC1 or CX_ADC2**.**

param1 is the address of the return value,

param2 is not used. Supposed to be always 0 (reserved for future use).

Note : the resolution and format of the return value depends on the platform (12 bits resolution, right aligned for STM32 platforms).

Example:

```
u32 ad1_value_pts;
u32 ad1_value;

CX_Configure( CX_ADC1, 0, 0 );

CX_Read( CX_ADC1, &ad1_value_pts, 0 );

// Convert values from points to mV
ad1_value = ( ad1_value_pts * VCC ) / 4096;

// Display the values
UTIL_uint2str( TextBuffer, ad1_value, 5, 0 );
DRAW_DisplayStringWithMode( 10, 50, TextBuffer, 5, NORMAL_TEXT, RIGHT );
```

Some platforms like STM32L owns only one ADC peripheral. In consequence, when Cx_ADC is used, all the CircleOS analog configuration is lost (touchscreen, Vbat measure...) until the Primer is reset and the CircleOS restarted.

13.7 Using the USART

Only the basic modes are handled by the CX functions. Synchronous modes (such as modes used for Smart cards) are not supported (the programmer has to configure the peripheral in his own application).

13.7.1 Configuration

int32 CX_Configure (CX_ID *what*, int32 *param1*, int32 *param2*)

what must be CX_USART

param1 is a pointer to a structure:

```
typedef struct
{
        uint32 BaudRate;        // An integer specifying the baud rate in bit per second.
        uint32 WordLength;      // The number of transferred data bit (either 8 or 9)
        uint32 StopBits;        // The number of stop bits (either 1 or 2)
        uint32 Parity;          // The parity : 0 for none, 1 for odd and 2 for even.
        uint32 HWControl        // The flow control: 0: none, 1: RTS, 2: CTS, 3: RTS and CTS
        uint8* RxBuffer;        // Rolling buffer to be used for reception
        uint32 RxBufferLen;     // Size of the receive buffer
        uint8* TxBuffer;        // Buffer to be used for transmission
        uint32 TxBufferLen;     // Size of the transmit buffer
}
tCX_USART_Config;
```

param2 is not used. Supposed to be always 0 (reserved for future use).

Example:

```
uint8 MyFifoRxBuffer[128];
uint8 MyFifoTxBuffer[128];
tCX_USART_Config my_UsartInit = {2400, 8, 1, 0, 0, MyFifoRx, 128, MyFifoTx, 128};
CX_Configure ( CX_USART, &my_UsartInit, 0 );
```

Note : USART uses a DMA controller and two FIFO buffers for Rx and Tx exchanges. Both buffers must be provided by the calling application, thanks to the addresses and length given to the "**CX_Configure**" function.

13.7.2 Receiving characters

int32 CX_Read (CX_ID *what*, int32 *param1*, int32 *param2*)

what is CX_USART

param1 is a pointer to a buffer,
param2 is a pointer to an integer that specifies:

1. the *size* of the buffer when calling the function,
2. the *number* of characters stored in the buffer when returning (*number* <= *size*).

Note that the value returned by the function could indicate an error such as:

- #define CX_NO_ERROR 0,
- #define CX_USART_PARITYERR 1,
- #define CX_USART_FRAMEERR 2,
- #define CX_USART_NOISEERR 4,

- #define CX_USART_OVERRUN 8.

CX_Read uses a circular FIFO managed by the DMA controller, but provided by the calling application.

Example:

```
      tCX_USART_Config my_UsartInit = { 9600, 8, 1, 0, 0, MyFifoRxBuffer, 128,
MyFifoTxBuffer, 128};
      uint8   buffer[32];
      uint32  ret, i, nb_byte = sizeof (buffer);

      CX_Configure (  CX_USART,  &my_UsartInit, 0);
      ret = CX_Read ( CX_USART, buffer, &nb_byte );

      if (ret)
      {
      // Error
      }
      else
      {
            if ( nb_byte ) //characters have been stored into buffer
            {
                  for ( i = 0 ; i < nb_byte ; i ++ )
                  {
                  // display buffer[i] on the monitor .
                  }
            }
```

13.7.3 Transmitting characters

int32 CX_Write (CX_ID *what*, int32 *param1*, int32 *param2*)

what is CX_USART

param1 is either a pointer to a buffer or zero.

When param1 is a non-zero value:
param2 is a pointer to an integer that specifies the *size* of the buffer to be sent. When returning, it indicates the number of sent bytes (supposed to be unchanged...).

When param1 is NULL:

param2 is a pointer to an integer. CX_Write will overwrite the contents of (*param2) by the number of characters that are still to be processed.

CX_Write uses a FIFO managed by the DMA controller, but provided by the calling application, and *param2 indicates the number of characters that are still buffered in this FIFO.

Note that the value returned by the function could indicate an error such as:

- #define CX_NO_ERROR 0,
- #define CX_USART_PARITYERR 1,
- #define CX_USART_FRAMEERR 2,
- #define CX_USART_NOISEERR 4,
- #define CX_USART_OVERRUN 8.

Example:

```
uint8 MyFifoRxBuffer[128];
uint8 MyFifoTxBuffer[128];
tCX_USART_Config my_UsartInit = { 2400, 8, 1, 0, 0, MyRxBuffer, 128, MyTxBuffer,
128};
uint32 nb_byteToSend = 12;
uint32 nb_byteSent = 0;
```

```
CX_Configure ( CX_USART, &my_UsartInit, 0);


// Send bufffer
nb_byteSent = nb_byteToSend;
CX_Write( CX_USART, "Hello World!", &nb_byteSent );

// Check the transmission
if ( nb_byteSent != nb_byteToSend )
      return ERROR;

// Wait for the end of the transfer
do
{
      CX_Write( CX_USART, 0, &nb_byteToSend );
}
while ( nb_byteToSend );

return OK;
```

13.8 Using the SPI

Only the basic modes are handled by the CX functions. I2S modes (for audio application) are not supported (the programmer has to configure the peripheral in his own application).

13.8.1 Configuration

int32 CX_Configure (CX_ID *what*, int32 *param1*, int32 *param2*)

what must be CX_SPI

param1 is a pointer to a structure:

```
typedef struct
{
	tCX_SPI_Speed Speed;		// The speed range of the serial bit rate.
	u8 WordLength;			// The number of transferred data bit. Standard is 8, but could
be 16 for some specific devices.
	u8 Mode;			// 1: master, 0: slave
	u8 Polarity;			// Indicates the steady state (idle state of the clock when no
transmission).
	u8 Phase;			// Phase: 0 indicates that the first edge of the clock when
leaving the idle state is active
					//	1 indicates that the second edge of the clock when
leaving the idle state is active
	u8 MSB1LSB0;			// First bit to be sent.  1: MSB first, 0: LSB first
	u8 Nss;				// Nss signal management : 1 = hardware / 0 = software
	u8* RxBuffer;			// Rolling buffer to be used for reception
	u32 RxBufferLen;		// Size of the receive buffer
	u8* TxBuffer;			// Buffer to be used for transmission
	u32 TxBufferLen;		// Size of the transmit buffer}
tCX_SPI_Config;
```

param2 is not used. Supposed to be always 0 (reserved for future use).

Useful provided enum and constants :

```
typedef enum
{
	CX_SPI_Speed_standard = 0,		// Equivalent to CX_SPI_Speed_Low (2)
	CX_SPI_Speed_VeryLow = 1,		// Typically 10kbps
	CX_SPI_Speed_Low = 2,			// Typically 100kbps
	CX_GPIO_Mode_High = 3,			// Typically 1Mbps
	CX_GPIO_Mode_VeryHigh = 4		// Typically 10Mbps
}
tCX_SPI_Speed ;
```

Note : the real speed depends on the CPU frequency.

```
#define CX_SPI_MODE_SLAVE		0
```

```
#define CX_SPI_MODE_MASTER      1
#define CX_SPI_POL_LOW          0
#define CX_SPI_POL_HIGH         1
#define CX_SPI_PHA_FIRST        0
#define CX_SPI_PHA_SECOND       1
#define CX_SPI_MSBFIRST         0
#define CX_SPI_LSBFIRST         1
#define CX_SPI_8_Bits           0
#define CX_SPI_16_Bits          1
#define CX_SPI_Soft             0
#define CX_SPI_Hard             1
```

Example:

```
uint8 MyFifoRxBuffer[128];
uint8 MyFifoTxBuffer[128];
tCX_SPI_Config my_SpiInit = {CX_GPIO_Mode_High, CX_SPI_8_Bits, CX_SPI_MODE_MASTER,
CX_SPI_POL_HIGH, CX_SPI_PHA_FIRST, CX_SPI_MSBFIRST, CX_SPI_Soft, MyFifoRx, 128, MyFifoTx,
128};

CX_Configure ( CX_SPI, &my_SpiInit, 0 );
```

Notes :

* SPI uses a DMA controller and two FIFO buffers for Rx and Tx exchanges. Both buffers must be provided by the calling application, thanks to the addresses and length given to the "**CX_Configure**" function.

* due to DMA configuration, when transmitting characters, unexpected characters are saved into the receive buffer. It is necessary to unstack these characters before the next read command. Take a look to the example below.

13.8.2 Receiving characters

int32 CX_Read (CX_ID *what*, int32 *param1*, int32 *param2*)

what is CX_SPI

param1 is a pointer to a buffer,
param2 is a pointer to an integer that specifies:

1.the *size* of the buffer when calling the function,

2.the *number* of characters stored in the buffer when returning (*number* <= *size*).

Note that the value returned by the function could indicate an error such as:

- #define CX_NO_ERROR 0,
- #define CX_SPI_OVERRUN 0x20
- #define CX_SPI_MODF 0x40.

CX_Read uses a circular FIFO managed by the DMA controller, but provided by the calling application.

Example:

```
uint8 MyFifoRxBuffer[128];
uint8 MyFifoTxBuffer[128];
tCX_SPI_Config my_SpiInit = {CX_GPIO_Mode_High, CX_SPI_8_Bits,
CX_SPI_MODE_MASTER, CX_SPI_POL_HIGH, CX_SPI_PHA_FIRST, CX_SPI_MSBFIRST,
CX_SPI_Soft, MyFifoRx, 128, MyFifoTx, 128};
```

```
CX_Configure ( CX_SPI, &my_SpiInit, 0 );
uint8   buffer[32];
uint32  ret, i, nb_byte = sizeof (buffer);

      CX_Configure (  CX_SPI,  &my_SpiInit, 0);
      ret = CX_Read ( CX_SPI, buffer, &nb_byte );

      if (ret)
      {
      // Error
      }
      else
      {
            if ( nb_byte ) //characters have been stored into buffer
            {
                  for ( i = 0 ; i < nb_byte ; i ++ )
                  {
                  // display buffer[i] on the monitor .
                  }
            }
```

13.8.3 Transmitting characters

int32 CX_Write (CX_ID *what*, int32 *param1*, int32 *param2*)

what is CX_SPI

param1 is either a pointer to a buffer or zero.

When param1 is a non-zero value:
param2 is a pointer to an integer that specifies the *size* of the buffer to be sent. When returning, it indicates the number of sent bytes (supposed to be unchanged...).

When param1 is NULL:

param2 is a pointer to an integer. CX_Write will overwrite the contents of (*param2) by the number of characters that are still to be processed.

CX_Write uses a FIFO managed by the DMA controller, but provided by the calling application, and *param2 indicates the number of characters that are still buffered in this FIFO.

Note that the value returned by the function could indicate an error such as:

- #define CX_NO_ERROR 0,
- #define CX_SPI_OVERRUN 0x20
- #define CX_SPI_MODF 0x40.

Example:

```
uint8 MyFifoRxBuffer[128];
uint8 MyFifoTxBuffer[128];
tCX_SPI_Config my_SpiInit = {CX_GPIO_Mode_High, CX_SPI_8_Bits,
CX_SPI_MODE_MASTER, CX_SPI_POL_HIGH, CX_SPI_PHA_FIRST, CX_SPI_MSBFIRST,
CX_SPI_Soft, MyFifoRx, 128, MyFifoTx, 128};
uint32 nb_byteToSend = 12;
uint32 nb_byteSent = 0;


CX_Configure ( CX_SPI, &my_SpiInit, 0);


// Send bufffer
nb_byteSent = nb_byteToSend;
CX_Write( CX_SPI, "Hello World!", &nb_byteSent );

// Check the transmission
if ( nb_byteSent != nb_byteToSend )
      return ERROR;
```

```
// Wait for the end of the transfer
do
{
      CX_Write( CX_SPI , 0, &nb_byteToSend );
}
while ( nb_byteToSend );

return OK;
```

Note : due to SPI functioning, to received characters it is necessary to write characters in order to generate the SPI clock. Take a look at the following example.

13.8.4 Example

SPI EEPROM read and write example, connected to the extension port.

```
#define EEPROM_SIZE 128
unsigned char EEPROM_Contents[EEPROM_SIZE];

//Write one word in EEPROM
int WriteEEPROMWord ( int addr32, unsigned val32 )
{
    const unsigned char Write_EEPROM_Operation = 0x40;
    const unsigned char EEPROM_address = 0;
    int nb_bytes = 1, nb_read  = 0;
    int i, ret = 0;
    char localbuf[(EEPROM_SIZE*sizeof(unsigned))+2];

    //Initialize tx buffer
    memset ( localbuf, 0xff, sizeof localbuf );
    localbuf[0] = Write_EEPROM_Operation;               //the write command
    localbuf[1] = addr32<<1;                            //address to be written
    localbuf[2] = (val32>>24) & 0xff;
    localbuf[3] = (val32>>16) & 0xff;
    localbuf[4] = (val32>>8) & 0xff;
    localbuf[5] = (val32>>0) & 0xff;
    CX_Write( CX_GPIO_PIN8, CX_GPIO_LOW, 0 );           //enable SPI (NSS)
    delay_50us();                                      //delay of 50us required
between NSS low and first clock edge
    nb_bytes = 6;
    CX_Write( CX_SPI, localbuf, &nb_bytes ); //Send operation (ID+address) + read
contents
    nb_bytes = 6;
    //we must read the dummy characters we generated
    while ( nb_read < 6 ) //Read as long all characters have been sent/received
    {
        nb_bytes = (6-nb_read);
        ret = CX_Read( CX_SPI, &(localbuf[nb_read]), &nb_bytes );
        nb_read+=nb_bytes;
    }

    EEPROM_Contents[ addr32 ] = val32;
    CX_Write( CX_GPIO_PIN8, CX_GPIO_HIGH, 0 ); //disable SPI (NSS)
}

int ReadEEPROM ( void )
{
    const unsigned char Read_EEPROM_Operation = 0x7F;
    const unsigned char EEPROM_address = 0;
    int nb_bytes = 1, nb_read  = 0;
    int i, ret = 0;
    char localbuf[(EEPROM_SIZE*sizeof(unsigned))+2];

    //Initialize tx buffer
    memset ( localbuf, 0xff, sizeof localbuf );
    localbuf[0] = Read_EEPROM_Operation;                    //the read command
```

```
    localbuf[1] = EEPROM_address;                           //first address to be
read

    CX_Write( CX_GPIO_PIN8, CX_GPIO_LOW, 0 );               //enable SPI (NSS)
    delay_50us();                                           //delay of 50us
required between NSS low and first clock edge
    nb_bytes = 2 + (EEPROM_SIZE*sizeof(unsigned));
    CX_Write( CX_SPI, localbuf, &nb_bytes );                //Send operation
(ID+address) + read contents

    while ( nb_read < ((EEPROM_SIZE*sizeof(unsigned))+2) )  //Read as long all
characters have been sent/received
    {
        nb_bytes = ((EEPROM_SIZE*sizeof(unsigned))+2 - nb_read);
        ret = CX_Read( CX_SPI, &(localbuf[nb_read]), &nb_bytes );
        nb_read+=nb_bytes;
    }
    memcpy ( EEPROM_Contents, &(localbuf[2]),
(EEPROM_SIZE*sizeof(unsigned)) );  //result is after the two first dummy bytes
    CX_Write( CX_GPIO_PIN8, CX_GPIO_HIGH, 0 );              //disable SPI

    return ret;

}
```

13.9 Using the I²C

Not yet implemented.

13.10 Examples

Take a look at the project example “**Cx Demo**” provided on the STM32 Circle web site, that implements all the peripherals for all Primer platforms.

http://www.stm32circle.com/

14. DMA2D (Chrom-ART Accelerator)

(EvoPrimer STM32F42x only)

14.1 Presentation

The DMA2D integration is the keystone to graphics potential of the STM32F429. Through a set of registers, it provides graphics management features that include:

- handling of bitmap files (up to 32 bits per pixel),
- management of image overlays, with or without transparency/blending,
- handling of object display to include multiple objects and their positions, sizes and layering (Z-order).

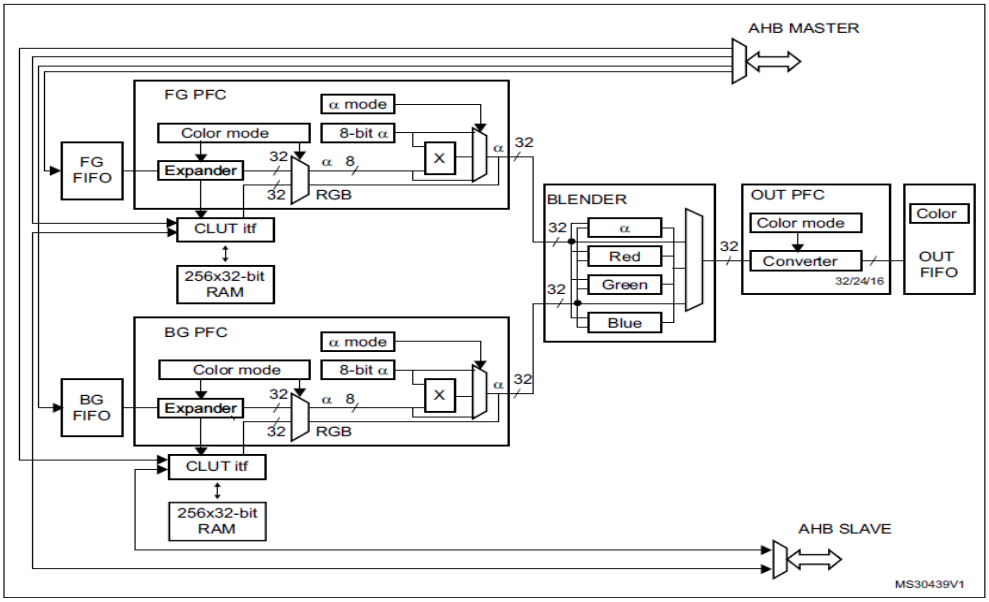
The CircleOS provides a specific API for easily implement these features by integration of the DMA2D peripheral and SDRAM driver.
This API covers:

- Handling of bit maps (display, overlaying and moving)
- Using transparency (pixel-level, global and blending)
- Dynamic versus Read-Only objects
- Management of objects and touch sensing
- Use of layering (Z-order)

14.2 DMA2D Integration in STM32F429 EvoPrimer

14.2.1 The DMA2D hardware

The following figure is taken from the STM32F429 reference manual. It shows the internal structure of the DMA2D and integration features that make the DMA2D particularly advantageous for managing displayed graphics.



These integrated features include:

- Two input pixel formats converters (PFC). These blocks are able to read and to decode the bitmaps files.
- The blender, which computes and mixes the data from the two input PFC.

- The ouput PFC, that decodes the information to be sent to the destination.
- The FIFOs used for both the inputs and the outputs connected to a specific DMA controller.

Integrating blending, encoding and decoding functionality in the microcontroller itself, relieves these requirements from the application and the core that would otherwise be required for this.

14.2.2 Large Internal Memory

The STM32F429 contains up to 2MB of internal FLASH.

In FLASH, a 320 x 240 pixel image requires a bit more than 75 K bytes when the colors are indexed in a 256-byte palette. Therefore, a few background images can be stored without problems.

14.2.3 SDRAM Controller

Storing bitmaps in FLASH is easy as long as there are not too many big pictures. But storing an editable object in RAM is more difficult to manage, since the size of the internal SRAM is always much smaller than the size of the internal FLASH.

Instead of embedding a very large internal SRAM, the new STM32F429 features an internal controller for an external SDRAM.

The STM32F429 EvoPrimer is equipped with 2Mb SDRAM, which can be used for large dynamic graphic objects.

14.2.4 Image of the Screen in RAM

The CircleOS version for STM32F429 EvoPrimer/Open4 kepts an image of the screen in the SDRAM. This is because it is not convenient to address the RAM of the LCD on these hardware platforms.

This image is sent (using the DMA) to the LCD when a DMA2D_ScreenRedraw has been executed.

This mode allows management of several screen images in SDRAM, and performing of rapid switches between these screens.

In the case of CircleOS, one Megabyte (half of the SDRAM) is reserved for the use of the DMA2D:

- Two screen images occupy 300 K bytes (2 x 320 x 240 x 16-bit pixels)
- The rest (724 K bytes) can be used for objects

14.3 Handling Bitmaps

14.3.1 Display of a picture

The DMA2D peripheral allows a bitmap file (the legacy .BMP format) to be loaded almost directly onto an LCD. The DMA2D does not decode the header of the bitmap. This must be managed by the application software, which initializes a set of registers that have been read from the header. The image is then displayed totally or partially on the LCD by the DMA2D.

The DMA2D manages most of the possible formats for bitmaps including: 8, 16, 24 and 32-bit, direct colors and indexed colors (using a palette).

In the C program, a bitmap must be declared as a byte array initialized by the contents of the binary file (translated into a .h test file). This example describes the code used to display the 640 by 320 pixel bitmap ("photo_x8 with 8-bit indexed color palette) shown here, on a display that is 320 by 240 pixels. Because the image is larger than the size of the LCD panel, it will be automatically cropped by the DMA2D API, as shown here.

Below is the code to declare for both this background image and a bird character (bird_A888) that is used later in the demonstration:

```
const unsigned char photo_x8[] =
            //A picture from a
camera...
    {
            //Indexed colors (8 bit
palette)
    0xff,0xff,
            //32-bit alignment
required for the palette
    #include
"image\background\photo640X320_x8.h"
      //The legacy .bmp file
    };

const unsigned char bird_A888[] =            //A small flying bird
    {                                        // .bmp file with 32 bit (A888)
colors.
    0xff,0xff,                               //for 32-bit alignment of the bitmap
data.
    #include "image\foreground\bird_A888.h"   //.bmp file contents
    };
```

To display the picture **photo_x8** as the background of a screen, all you have to do is to create an object, create a link between the object and the bitmap, then to make the bitmap visible.

```
    BG_Image = DMA2D_ObjectCreate(640,320,1);       //Create an empty object
    DMA2D_ObjectAssignBitmap(BG_Image,photo_x8,0);  //Link bitmap and object
    DMA2D_ObjectSetToScreen(BG_Image, SCREEN_ID);   //Attach the object to a screen
    DMA2D_ObjectSetVisible(BG_Image, 1);            //Make the object visible
    DMA2D_ScreenRedraw(SCREEN_ID);                  //and repaint the whole screen.
```

The palette for the bitmap is loaded automatically by the DMA2D, based on the header information that is decoded by the application software (in this case the API).

The last command (DMA2D_ScreenRedraw) scans the list of objects assigned to a screen and repaints all the visible ones. Note that the size of the background image (640x420 pixels in this example) can be bigger than the size of the screen (320x240 for our demonstration platform).

The bitmap can be positioned based on provided coordinates. 0 for X and Y axes are the lower left had corner of the LCD. To fill the entire display area of the LCD, the coordinates X and Y must be negative (or equal to 0). Positive coordinates will leave an area that has not been drawn starting in the lower left corner of the display to the coordinate where the lower left corner of the background image is displayed.

If you want to center the image, you can "move" the background before redrawing:

```
    DMA2D_ObjectMove(BG_Image, PosX_BG = -160, PosY_BG = -120 );  //set the position
```

Because the overall object is too big to be displayed in our screen, it is automatically cropped at the borders of the LCD panel.

14.3.2 Overlaying Bitmaps

The DMA2D allows other objects to be placed on the background. Overlaying objects on a background or on other objects is particularly advantageous with the DMA2D because only the areas affected by a modification are redrawn. Without the DMA2D, changing the position of a displayed object requires either pixel-by-pixel redrawing of the full display, or complex application code (and a lot of processor time) to manage this functionality in the same manner as the DMA2D.

With our STM32F429, the DMA2D peripheral writes only the affected pixels. So, to add an object over this background, we simply create it, set the position (relative to the display and not the background) and make it visible:

```
    int x_bird = 100, y_bird = 100;
    FG_Bird = DMA2D_ObjectCreate(48, 48 , 1);

    DMA2D_ObjectAssignBitmap(FG_Bird, bird_A888,
0);
    DMA2D_ObjectSetToScreen(FG_Bird, SCREEN_ID);
    DMA2D_ObjectSetVisible(FG_Bird, 1);
    DMA2D_ObjectMove(FG_Bird, x_bird, y_bird);
```

14.3.3 Moving Objects

To make an object move through the display (for example to make our bird fly), it is only necessary to periodically recall the last line with a new, different position (x_bird, y_bird):

```
    DMA2D_ObjectMove(FG_Bird, x_bird++, y_bird++); // the bird moves to up-right.
  ...
    DMA2D_ScreenRedraw( SCREEN_ID);                 //and repaint the whole screen.
```

The DMA2D hardware peripheral is addressed only when calling the DMA2D_ScreenRedraw function. The rest of the API functions just configure the descriptions of the objects that will be used by the DMA2D.

Note that it is possible to animate the flying bird by managing several bitmaps, each showing the bird in a different state of movement. Then we periodically change the bitmap assignment. This is how the butterfly is handled in the demonstration:

```
    DMA2D_ObjectAssignBitmap(FG_Bird, bird_fly_A888[n], 0);
  ...
    DMA2D_ScreenRedraw(SCREEN_ID);                  //and repaint the whole screen.
```

Thanks to the integration of the DMA2D peripheral, the display changes that allow us to animate movement are represented by just a few lines of simple code.

14.4 Transparency

14.4.1 Alpha component for the pixel

Transparency of an object that is overlayed on the background is handled at different levels.

First, transparency may be native to the original bitmap. A 32-bit bitmap file contains 8-bit transparency information for each pixel (24-bit color plus an 8-bit alpha). The transparency is also referred to as the "alpha" component. If you choose a file in a 32-bit format for display using the DMA2D, the original transparency for each pixel is managed but the DMA2D. If you choose any other, smaller format, the alpha constant is not defined in the bitmap and therefore is assumed to be equal to 255 (an opaque object) when displayed using the DMA2D.

In addition to the native transparency of a specific bitmap, the DMA2D can be used to manage transparency according to a global alpha constant.

14.4.2 Global alpha constant

Global alpha constant is used to define the global transparency of an object when it is displayed. If an object has both the original bitmap transparency and a global transparency, the resulting alpha value for each pixel will be the combination of the two parameters.
The two alpha values are multiplied and the result divided by 256:

$$alpha_{out} = \frac{(alpha_{pixel} * alpha_{global})}{256}$$

This means that a fully transparent (invisible) pixel in the bitmap (alpha = 0 in the bitmap file) remains invisible regardless of the global transparency of the object.

14.4.3 Blending

This example shows how alpha factors are combined when objects are overlayed. The DMA2D allows complex combinations in blending, which are not explored here.

For the purpose of this demonstration, each time an object is displayed, the API assumes that the "current background" is opaque (alpha = 255). "Current background" is the term that we use for the result of the display of all the previous layers/objects.

The final pixel is the result of the combination of all the alpha values of the original bitmaps, weighted by the global alpha values of each object. Note that, if the object at the top level (the highest z-order) is opaque (alpha=255 for both the object and the pixel), this object is displayed without any interference from the lower objects.

The following code shows how to create, in an editable object, a fully transparent part, and a opaque part that will be globally tunable by the global transparency ratio:

```
DMA2D_ObjectSetTransparency(FG_Text,0); //Full transparency for the next command
LCD_FillRect( 0,0,220,80,RGB_RED);      //Fill as transparent the whole object
DMA2D_ObjectSetTransparency(FG_Text,0xff);  //The elliptic surface will be opaque
DRAW_Ellipse( 110,30,110,35,RGB_RED,RGB_RED,1,1);  //Draw a centered opaque ellipse
DMA2D_ObjectSetTransparency(FG_Text,0x80);  //Set the global transparency
```

14.5 Editable objects

14.5.1 Dynamic vs Read Only Objects

Up to this point, we have only handled graphics that were edited on a PC and saved in BMP format. The bitmap was included as a ".h" file into the FLASH that is treated by the application as read only memory.

The API allows objects to be handled in RAM as well. These objects can be dynamically edited by the application. In the image at the right, the text is written directly onto a dynamic object. The red ellipse itself has been drawn using the standard library functions as shown in this code:


```
  FG_Text = DMA2D_ObjectCreate(220,80 ,0);      // The 3rd parameter means "read-
only==0"
  DMA2D_ObjectSetToScreen(FG_Text, SCREEN_ID);

  //All the following commands will be applied to the object instead of the screen
  DMA2D_ObjectSelect(FG_Text);
  DMA2D_ObjectSetTransparency(FG_Text,0);
  LCD_FillRect( 0, 0, 220, 80, RGB_RED);
  DMA2D_ObjectSetTransparency(FG_Text,0xff);
  DRAW_Ellipse(110,30,110,35,RGB_RED, RGB_RED, 1, 1);
  DRAW_SetBGndColor(RGB_RED);                   // Background color for the text
  DRAW_SetTextColor(RGB_WHITE);
  DRAW_DisplayStringWithMode( 60, 45, "Dynamic", 0, NORMAL_TEXT, LEFT);
  DRAW_DisplayStringWithMode( 57, 30, "layered", 0, NORMAL_TEXT, LEFT);
  DRAW_DisplayStringWithMode( 50, 15, "graphics", 0, NORMAL_TEXT, LEFT);
```

14.5.2 Editing a dynamic object

The DRAW commands shown above typically pertain to the whole screen. The coordinates x = 0, y = 0 are the coordinates of the lower left corner of the LCD. After calling DMA2D_ObjectSelect(), these DRAW_xxx commands are assigned to the currently selected object. The coordinates are considered relative to the object and (0,0) matches with the lower left corner of the rectangle containing the object.

When a dynamic object is declared, a move command is applied to the whole object. This means that the object will move with its contents (the ellipse and the text).

Note that the object is always stored in RAM as a 32 bit image. Each pixel has a 24-bit color and a 8-bit alpha component.

14.6 Advanced features

14.6.1 Z-Order

The objects are sorted to define the order of drawing. They are stored in an array, and the index in this array is named Z-order. Graphically, the Z-order specifies the relative position of the object layer (eg. Its position on the Z-axis). "Z-order == 0" corresponds to the first drawn object (typically the background).

The API provides functions to increment/decrement the objects Z-order. Two others functions allow placement of the object at the upper layer, or as the background.

```
DMA2D_ObjectPushZ(Bird2_Image, SCREEN_SLIDE);
```

14.6.2 Touchscreen and objects

When implementing objects in a touchscreen interface, the function DMA2D_ObjectFind returns a pointer to the object that is the owner of the pixel located at (x,y). When several objects are layered at the same position, the upper most layer is considered.

So, when the touchscreen is pressed, it's quite simple to detect which object has been touched:

```
    if (  TOUCHSCR_IsPressed() )
        {
        posx = TOUCHSCR_GetPosX();
        posy = TOUCHSCR_GetPosY();
        SelectedObject = DMA2D_ObjectFind(CurrentScreen, posx, posy );
        if ( SelectedObject  == FG_Bird )
            {
            bird_hit = 1;

             ....
```

14.7 Files

DMA2D_Graphic.c : specific DMA2D management.

lcd_spec.c : low level specific functions

This modules provides useful functions for DMA2D management, that means use of the embedded DMA2D peripheral of the STM32F429x micro-controller.

14.8 Functions

14.8.1 Internal functions

In order to reduce the API list, one enter point is used :

u32 DRAW_DMA2D_Dispatch(u32 fct, u32 param1, u32 param2, u32 param3)

The first parameter is a pointer to the specific function requested, and the last three parameters are passed to the specific function.

void DMA_LCD_UpdateScreen()

Refresh the LCD with the screen image contained in the SDRAM (screen = number of screen to display, 0 or 1) by copying the RAM screen area to the LCD controller with standard DMA.

For optimization, only modified lines are refreshed.

This function is called every systick, by the DRAW handler.

14.8.2 API's

void DMA2D_ScreenInit()

Initializes all screen memory:

- clear all screens,
- clear all the objects,
- selects screen 0,
- display the main EvoPrimer screen,
- clear all application objects.

void DMA2D_Update()

Not yet implemented. Actually calls the *DMA2D_ScreenInit* function.

void DMA2D_ScreenClear(screen, del)

Clear a screen image contents (screen = number of screen to clear, 0 or 1), by hiding all objects of the screen. If del = 1 all objects are deleted (not yet implemented).

void DMA2D_ScreenSelect(screen,mode)

Activates one of the 2 screen RAM area.

- screen = number of screen to activates (0 or 1),
- mode = activation mode (SELADDR (1) = the following accesses will write or read to/from this screen, SELDISP (2) = this screen is the displayed screen, SELFULL (3) the screen is selected for RAM access and display.

void DMA2D_ScreenSetDirty(screen)

Declares that the screen is “dirty” that means that it is necessary to refresh its display.
Screen = number of screen to activates (0 or 1)

void DMA2D_ScreenRedraw(screen)

Rebuild in memory the contents of the screen from the object list.

void DMA2D_ScreenCopy(dest,src)

Copy a screen into another.

- dest : the number of the destination screen.

- src : the number of the source screen.

tDMA2D_pObject DMA2D_ObjectCreate(width,height,rdonly)

Initializes the object structure

Allocates the graphic area in RAM when rdonly = 0

Reset memory and position.

Return a pointer on the object structure (zero if failed)

If rdonly = 1, the bitmap image is expected to be located in FLASH memory.

void DMA2D_ObjectDelete(obj)

Set the object pointed by obj as deleted

Remove the object from all screens and mark as dirty these screens.

Free the memory of the object (not yet implemented, because no garbage collector yet implemented).

int DMA2D_ObjectSetToScreen(obj,screen)

Attach an object to the specified screen.

Increment the object number

Reset the position of the object.

int DMA2D_ObjectAssignBitmap(obj,pbitmap,fbitmap0constant1)

Assign a bitmap to an object.

- obj : object to be added on the screen,
- pbitmap pointer to the bitmap (flash area containing the BMP file image),
- fbitmap0constant1 : bitmap format (0 = BMP file, 1 = colored rectangle).

Returns 0 if OK, -1 if format not supported.

Note :bitmaps with of 4, 8, 16 or 32 bits format are supported.

int DMA2D_ObjectSetPartial(obj,width,height)

Resize an object. This function is used with DMA2D_ObjectSetOffset to display just a part of an object.

Returns 0 if OK, -1 if bad parameters.

int DMA2D_ObjectSetOffset(obj,x,y)

Specify an offset (from the low left corner) for the display of an object. This function is used to reduce the part of the image that will be displayed. It has to be combined with a call to DMA2D_ObjectSetPartial.

Returns 0 if OK, -1 if bad parameters.

int DMA2D_ObjectMove(obj,x,y)

Move an object for the selected screen, to the x,y coordinates.

void DMA2D_ObjectSetVisible(obj,mode)

Make an object visible/not visible.

1 = visible, 0 is hidden.

int DMA2D_ObjectSelect(obj)

Select the current object, on which the following drawing commands will be done (dynamic objects, that should be created with no read-only parameter).

The DRAW_xx functions can be applied either to a screen or to a simple object. DMA2D_ObjectSelect select an object, and DMA2D_ScreenSelect will unselect it. It could be also unselected by selecting another object.

`void DMA2D_ObjectSetTransparency(obj,transp)`

Modify the transparency for the current object. The DRAW_ functions will agglomerate this transparency information to build the 32-bit pixel.

Transparency: = new global alpha constant for the object (0 to 255).

`color32_t DMA2D_ObjectGetPixel(obj, x, y)`

Get a pixel within an object

Return the 32bits value of the pixel pointed by the x,y coordinates. (returns 0 if failed)

`u32 DMA2D_ObjectGetIntersection(screen,obj1, obj2)`

Find the intersection between two objects in a specified screen.

Ignore the pixel when one is fully transparent.

The return value quantifies the intensity of the intersection on a 32 bit number (0 if not intersection).

`int DMA2D_ObjectGetPos(obj, screen)`

Returns the current position of an object in the specified screen.

Returns a 32 bit value : the upper 16 bit half-word contains Y, and the lower 16-bit contains X.

`int DMA2D_ObjectGetSize(obj)`

Returns the size of an object.

Returns a 32 bit value. the upper 16 bit half-word contains the height, and the lower 16-bit contains the width.

`tDMA2D_pObject DMA2D_ObjectFind(screen,x,y)`

Search for an object (upper Z-order) that has a visible pixel on the point specified by the coordinates (x,y).

Retunrs 0 if failed.

`u32 DMA2D_ObjectGetImage(obj)`

Returns a pointer to the object image.

`int DMA2D_ObjectPushZ(obj, screen)`

Push the object to one layer deeper (heading to the background).

Returns the current Z order after the operation (-1 if the object is not assigned to the screen.).

`int DMA2D_ObjectPopZ(obj, screen)`

Pop the object to one layer upper (heading to the foreground).

Returns the current Z order after the operation. (-1 if the object is not assigned to the screen).

`int DMA2D_ObjectSetForeground(obj, screen)`

Pop the object to the top layer (the foreground).

Returns the current Z order after the operation. (-1 if the object is not assigned to the screen).

`int DMA2D_ObjectSetBackground(obj, screen)`

Push the object to layer 0 (the background).

Returns the current Z order after the operation. (-1 if the object is not assigned to the screen).

`void DMA2D_SetTransform( tFctXY fct_X, tFctXY fct_Y )`

Defines the transform function before calling DMA2D_ObjectTransform

Example :

DMA2D_SetTransform(fRot90X, fRot90Y);

`u32 DMA2D_ObjectTransform( tDMA2D_pObject obj_dest, tDMA2D_pObject obj_src, tTrans* trsf )`

Generic transform function used to copy, rotate, resize an object. The transformation to use should be defined previously by calling DMA2D_SetTransform.

This function takes every pixel of the *SOURCE*, and calculates its new value.

Example :

```
int fRot90X( int x, int y ) { return y; }
int fRot90Y( int x, int y ) { return 48-x-1; }
```

`u32 DMA2D_ObjectTransformReverse( tDMA2D_pObject obj_dest, tDMA2D_pObject obj_src, tFctObjXY fct )`

Generic transform function used to copy, rotate, resize, modify the color of an object....

This function takes every pixel of the *DESTINATION*, and calculates its new value.

Example:

DMA2D_ObjectTransformReverse(obj_dest, obj_src, ColorInvertRB);

where ColorInvertRB inverts red and blue colors of the object.

14.8.3 Structures and variables

Object structure :

```
tDMA2D_Object:
        {
        // Current position on screen
        short int mX[MAX_DMA2D_SCREEN], mY[MAX_DMA2D_SCREEN];

        // z_order (index) for each screen where the object is used.
        short int mZOrder[MAX_DMA2D_SCREEN];

        // Characteristics of the object
        struct {
                int mIsVisible: 1;  // 0: not visible, 1: visible
                int mIsRdonly:1;  // 1 if FLASH, 0 if RAM
                int mIsPartial: 1;  //  ( (width_bitmap!= width_used) || (height_bitmap!= height_used) )
                int mIsDirty:    1;  // indicate that the object has been modified. Must be redrawn
                int mIsCst:      1;  // the object is a simple rectangle filled with a constant color.
                int mFormat:   8;  // pixel format.
                unsigned  mAlpha: 8;  // global alpha constant.
                int mScreenCnt:     3;  // counter to define the number of screen that use this object
                int mIsDeleted:       1;  // the object has been deleted.
                }   mStatus;
```

```
short int mCurrentRect_x;                // x coordinate when a DRAW command is used for a selected object
short int mCurrentRect_y;                // y coordinate when a DRAW command is used for a selected object
short int mCurrentRect_width;            // width when a DRAW command is used for a selected object
short int mCurrentRect_height;           // height when a DRAW command is used for a selected object
short int mCurrentRect_relpos_x;         // current x coordinate when a rectangle is filled from the selected object
short int mCurrentRect_relpos_y;         // current y coordinate when a rectangle is filled from the selected object
short int mCurrentRect_transparency;     // global transparency for the selected object

//Source of the object
short int       mWidthTotal;     // width of the bitmap before any resizing
short int       mHeightTotal;    // height of the bitmap before any resizing

//Part definition
short int       mWidthUsed;      // subset of the bitmap to be used
short int       mHeightUsed;     // subset of the bitmap to be used
short int       mOffsX;          // X offset for subset
short int       mOffsY;          // Y offset for subset

union {
        color32_t *   mBitmap;       // pointer to the image (with Alpha)
        color32_t     mARGB_Cst;  // Constant ARGB 32 bit color (with Alpha)
        } mData;                     // mARGB is used for the register mode (to fill a rectangle with a constant).

//Memory management. The object are created in
void*       mNextObject;        // pointer to the next object in the memory pool.
unsigned mSize;                 // object size = number of bytes
unsigned mSizeGraph;            // object size = number of pixels
union
        {
        color32_t           mDynamicImage[1];   // Pointer to an object in RAM
        tBmpDescriptor      mBmpDesc;           // Bitmap descriptor used for readonly bitmap
} mContents;
```

Screen structure

```
tDMA2D_Screen :
        {
        struct   {
                int mIsSelected:   1;     // Current screen displaid on the LCD
                int mIsDirty:      1;     // Contents has been modified (would need an update)
                int mWantUpdate:1;        // Wait for an update...
```

```
        int mOrientation:   2;     // Current orientation of the screen
        } mStatus;

    // Object to be displayed in the screen. The index matches with the Z-order
    tDMA2D_pObject  mObjectTable [MAX_DMA2D_OBJECT];

    int mObjectCount;               // Number of objects used in the table above.

    Union   {

        // The memory is seen horizontaly (a set of rows)
        tpixel24 mVerView  [PHYS_SCREEN_HEIGHT] [PHYS_SCREEN_WIDTH];

        / The memory is seen verticaly (a set of columns)
        tpixel24 mHorView  [PHYS_SCREEN_WIDTH]  [PHYS_SCREEN_HEIGHT];

        // The memory is seen as a continuous global set of pixels
        tpixel24 mArray    [PHYS_SCREEN_HEIGHT * PHYS_SCREEN_WIDTH];

        } mScreenImage;             // The set of pixels
    };
```

Screen table (2 screens declared into the SDRAM) :
tDMA2D_Screen LCD_ScreenImage [MAX_DMA2D_SCREEN];

Memory pool = the memory pool used for the object allocation :
unsigned char ObjectSingleMemoryPool [OBJECT_SINGLE_MEMORY_POOL_SIZE];

Transformation structures

```
typedef int ( *tFctXY )( int x, int y );
tTrans :
    {
    tFctXY  mFctX;
    tFctXY  mFctY;
    u32     mDefaultColor;
    };
```

Screen access mode
enum SELECT_MODE { SELNONE = 0, SELADDR = 1, SELDISP = 2, SELFULL = 3 };

Max items

```
#define MAX_DMA2D_SCREEN            2       // Number of screens stored in SDRAM
#define MAX_DMA2D_OBJECT            256     // Number of object managed per screen
```

14.9 Examples

Take a look at the project example “**DMA2DEMO**”, “**SPACE 2D**” and “**DMA2D_TR**” provided on the STM32 Circle web site.

http://www.stm32circle.com/

15. Backup registers

Several STM32 registers are saved by the battery when the Primer is powered off (number is depending on the STM32 type).

For STM8L platform, backup registers are simulated into the EEPROM data area.

Some registers are reserved for CircleOS, some for applications (7 to 10).

15.1 Registers saved at power off

Number	Label	Contents	Values
1	SYS1	Current application	
2	SYS2	Several system informations	
3	SYS3	BKP_BKLIGHT = LCD contrast value	0x1000, 0x4000, 0x8000, 0xC000, 0xFFFF0
4	SYS4	Font menu	0 = standard font x1 1 = standard font x 2 2 = medium font
5	SYS5	Audio_Volume	0-255 (0 to -127 db attenuation)
6	SYS6	Spare	
7	USER1	Free for application	
8	USER2	Free for application	
9	USER3	Free for application	
10	USER4	Free for application	
11	SYS7	Touchscreen informations	
12	SYS8	Touchscreen calibration points	BKP_TS_X0
13	SYS9	Touchscreen calibration points	BKP_TS_Y0
14	SYS10	Touchscreen calibration points	BKP_TS_X1
15	SYS11	Touchscreen calibration points	BKP_TS_Y1
16	SYS12	Touchscreen calibration points	BKP_TS_X2
17	SYS13	Touchscreen calibration points	BKP_TS_Y2

15.2 SYS2 register details

Bit	Contents	Values
0:2	CPU speed	1 to 5
3	Speaker	ON/OFF
4	Joystick	ON/OFF
5	MEMS	ON/OFF
6	Audio mute	ON/OFF
7	Autorun	ON/OFF
8	Touchscreen as only Input	ON/OFF
9	Buzzer	ON/OFF
10:15	Spare	

16. Glossary

Term	Description
API	Application Programming Interface : a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications
OS	Operating System
MEMS	Micro Electro Mechanical System : an integrated accelerometer, in this case

Alphabetical Index

APP_FindAppIndex(index).....54
APP_LaunchAppli(index).....54
AUDIO_BUZZER_SetToneFrequency.....34
AUDIO_CODEC_CRs.....32
AUDIO_Cpy_Mono().....34
AUDIO_Dec_Volume(dB number).....35
AUDIO_DeviceSoftwareReset.....34
AUDIO_GetMode().....35
AUDIO_Handler().....33
AUDIO_I2C_Init().....33
AUDIO_I2C_Read_Register(register to read)…34
AUDIO_I2C_ReadMultiByte.....34
AUDIO_I2C_Write_register(register number)....34
AUDIO_I2C_WriteMultiByte.....34
AUDIO_Inc_Volume(dB number).....35
AUDIO_Init_Audio(length, frequence, format)…33
AUDIO_Init_Voice(length, frequence, format)…33
AUDIO_Init().....33
AUDIO_IsMute().....35
AUDIO_MUTE(ON/ OFF).....35
AUDIO_Play.....32
AUDIO_Play(buffer, size).....35
AUDIO_Playback_GetStatus().....35
AUDIO_Playback_Stop().....35
AUDIO_PlaybackBuffer_GetStatus(value).....35
AUDIO_ReadRegister(register_to_read, data_to_read).....36
AUDIO_ReadRegister(register_to_read).....36
AUDIO_Record.....32
AUDIO_Record_Buffer_GetStatus(value).....35
AUDIO_Record_Stop.....35
AUDIO_Record(buffer, size).....35
AUDIO_Recording_GetStatus().....35
AUDIO_Set_Volume().....34
AUDIO_SetMode(new mode).....35
AUDIO_Shutdown().....33
AUDIO_SPEAKER_OnOff(ON/OFF).....35
AUDIO_Welcome_Msg().....33
BUTTON_GetMode().....43
BUTTON_GetState().....43
BUTTON_SetMode(mode).....43
BUTTON_WaitForRelease().....43
CX_Configure(tCX_ID what, int32 param1, int32 param2).....75
CX_Read(tCX_ID what, int32 param1, int32 param2).....75
CX_Write(tCX_ID what, int32 param1, int32 param2).....75
DB_LED_Init().....54
DefaultAction().....42
DMA_LCD_UpdateScreen().....95
DMA2D_ObjectAssignBitmap(obj,pbitmap,fbitmap0constant1).....96
DMA2D_ObjectCreate(width,height,rdonly).....96
DMA2D_ObjectDelete(obj).....96
DMA2D_ObjectFind(screen,x,y).....97
DMA2D_ObjectGetImage(obj).....97
DMA2D_ObjectGetIntersection(screen,obj1, obj2) 97
DMA2D_ObjectGetPixel(obj, x, y).....97
DMA2D_ObjectGetPos(obj, screen).....97
DMA2D_ObjectGetSize(obj).....97
DMA2D_ObjectMove(obj,x,y).....96
DMA2D_ObjectPopZ(obj, screen).....97
DMA2D_ObjectPushZ(obj, screen).....97
DMA2D_ObjectSelect(obj).....96
DMA2D_ObjectSetBackground(obj, screen).....97
DMA2D_ObjectSetForeground(obj, screen).....97
DMA2D_ObjectSetOffset(obj,x,y).....96
DMA2D_ObjectSetPartial(obj,width,height).....96
DMA2D_ObjectSetToScreen(obj,screen).....96
DMA2D_ObjectSetTransparency(obj,transp)….97
DMA2D_ObjectSetVisible(obj,mode).....96
DMA2D_ObjectTransform(tDMA2D_pObject obj_dest, tDMA2D_pObject obj_src, tTrans* trsf).....98
DMA2D_ScreenClear(screen, del).....95
DMA2D_ScreenCopy(dest,src).....95
DMA2D_ScreenInit().....95
DMA2D_ScreenRedraw(screen).....95
DMA2D_ScreenSelect(screen,mode).....95
DMA2D_ScreenSetDirty(screen).....95
DMA2D_SetTransform(tFctXY fct_X, tFctXY fct_Y).....98
DMA2D_Update().....95
faboutIni().....53
fAboutMgr().....53
fCalibration().....53
fconfig().....53
FctAppIni().....54
FctAppMgr().....54
fHighSound().....42
fInBoth().....53
fInJoystick().....53
fInMems().....53
fInTchscr().....53
fInterface().....53
flagWrite_AUDIO_CODEC_CRs.....32
flowSound().....42
fMuteSoundf().....42
fno().....52
fPower().....53
fQuit().....52
FS_Close(PFILEINFO fileinfo).....60
FS_Delete(PVOLINFO volinfo, u8 *path).....60
FS_Explorer_Ini().....61
FS_Explorer().....61
FS_GetNextEntry(PVOLINFO volinfo, PDIRINFO dirinfo, PDIRENT dirent).....61
FS_GetPathFilter().....61
FS_GetSDCardCurrentPath().....61
FS_GetSDCardVolInfo().....61
FS_GetVolumeInfo(u8 unit, u32 startsector, PVOLINFO volinfo).....61

FS_Mount((enum STORAGE_device device). .60
FS_OpenDirectory(PVOLINFO volinfo, u8 *dirname, PDIRINFO dirinfo)....61
FS_OpenFile(PVOLINFO volinfo, u8 *path, u8 mode, PFILEINFO fileinfo)....60
FS_ReadFile(PFILEINFO fileinfo, u8 *buffer, u32 *successcount, u32 len) 60
FS_Seek(PFILEINFO fileinfo, u32 offset)....61
FS_SetPathFilter(u8 *filter)....61
FS_Unmount(enum STORAGE_device device) 60
FS_WriteFile (PFILEINFO fileinfo, u8 *buffer, u32 *successcount, u32 len)....60
fSDCard()....52
fSetAutorun()....53
fSetBacklight_Mgr()....52
fSetBeep()....53
fSetInput()....53
fSetMenu_Ini()....53
fSetMenu_Mgr....53
fSetPllRange_Mgr()....52
fSetSpeaker()....53
fSetTime_Ini()....52
fSetTime_Mgr()....52
fShutdown()....52
fTest_Mgr()....53
fyes()....52
I2S2....32
I2S3....32
InitListDMA()....48
IS_PLAYING....32
IS_RECORDING....32
JOYSTICK_CircularPermutation(abs_direction, iter)....43
JOYSTICK_GetState()....44
JOYSTICK_Handler()....43
JOYSTICK_WaitForRelease()....44
LCD_ChangeFont(enum ENUM_FontID ID)...30
LCD_GetTransparency()....30
LCD_SetFontDef(tFontDef* FontDef)....30
LCD_SetTransparency(u8 NewTransparency)30
LIST_DetectMove....48
LIST_DRAW_SetImage....48
LIST_GetNewSelectedItem()....48
LIST_LCD_RectRead(x, y, width, height)....48
LIST_Manager()....49
LIST_RefreshItem....48
LIST_Set....49
LIST_StoreChar....48
LIST_StoreString....48
MedianFilter()....38
MENU_ClearCurrentCommand()....54
MENU_ClearCurrentMenu()....55
MENU_GetBGndColor()....54
MENU_GetTextColor()....54
MENU_Handler()....52
MENU_Print(str)....54
MENU_Question(@str,answer)....54
MENU_Quit()....54
MENU_Remove()....54
MENU_RestoreAppliDivider()....55
MENU_Set(@tMenu)....54
MENU_SetAppliDivider(divider)....55
MENU_SetBGndColor(color_t)....54
MENU_SetLevel_Ini()....55
MENU_SetLevel_Mgr(@value,@value_range). 55
MENU_SetLevelTitle(@title)....54
MENU_SetTextColor(color_t)....54
ObjectTransformReverse(tDMA2D_pObject obj_dest, tDMA2D_pObject obj_src, tFctObjXY fct)....98
POWER_Handler()....46
POWER_Init()....46
PWR_RESET_TIME....46
PWR_SET_TIME....46
PWR_STATE_CHARGING....45
PWR_STATE_EMPTY....45
PWR_STATE_FULL....45
PWR_STATE_LOW....45
PWR_STATE_NOBAT....45
PWR_STATE_NORMAL....45
PWR_STATE_UNDEF....45
RefreshItem(sel,isInverted)....52
SET_FLAG_WRITE_CODEC_CRS(x)....32
SHUTDOWN_Action()....46
tFontDef....31
tFontTable....31
tList()....49
tListItem()....49
tMenu()....55
tmenuItem()....55
TOOLBAR_Button....41
TOOLBAR_ChangeButton....42
TOOLBAR_Handler()....41
TOOLBAR_Init()....41
TOOLBAR_Redraw....41
TOOLBAR_SelectButton(button)....41
TOOLBAR_Set(@ new toolbar)....42
TOOLBAR_SetDefaultToolbar....42
TOOLBAR_UnSelectButton(button)....41
TOUCHSCR_Cal....39
TOUCHSCR_CalculateCalibration()....38
TOUCHSCR_GetAbsPos()....38
TOUCHSCR_GetMode....38
TOUCHSCR_GetPos()....38
TOUCHSCR_Handler()....37
TOUCHSCR_Info....38
TOUCHSCR_Init()....37
TOUCHSCR_IsPressed()....38
TOUCHSCR_SetMode()....37
TOUCHSCR_SetSensibility()....38
TOUCHSCREEN_Calibration()....37
TOUCHSCREEN_Drawing()....38
tPOINT....39
tToolbar_Item()....42
tToolbar()....42
u32 FS_Eof(PFILEINFO fileinfo)....63
u32 FS_FileCmp(char* fn1, char* fn2)....63

u32 FS_FileCopy(char* dest, char* src).....63
u32 FS_Gets(char* ptr, int len, PFILEINFO fileinfo).....63
u32 FS_Size(PFILEINFO fileinfo).....63
u32 FS_Tell(PFILEINFO fileinfo).....63
UTIL_GetAppAddress (const u8 *AppName)….65
UTIL_GetBat (void).....65
UTIL_GetBatStatus().....65
UTIL_GetDividerHandler(enum eSchHandler Idx) 65
UTIL_GetIrqHandler (s32 Offs).....65
UTIL_GetPll (void).....64
UTIL_GetPrimerType (void).....64
UTIL_GetSchHandler (enum eSchHandler Ix). .64
UTIL_GetTemp (void).....65
UTIL_GetVersion (void).....64
UTIL_int2str (u8 *ptr, int_t X, len_t digit, bool fillwithzero).....66
UTIL_IsStandAloneMode (void).....64
UTIL_ReadBackupRegister (index_t BKP_DR).65
UTIL_SaveScreenBMP().....66
UTIL_SetDividerHandler(enum eSchHandler IDx, u16 divider).....64
UTIL_SetIrqHandler (s32 Offs, tHandler pHDL) 65
UTIL_SetPll (enum eSpeed speed).....64
UTIL_SetSchHandler (enum eSchHandler Ix, tHandler pHDL).....64
UTIL_SetTempMode (bool mode).....65
UTIL_uint2str (u8 *ptr, uint_t X, len_t digit, bool fillwithzero).....65
UTIL_WriteBackupRegister (index_t BKP_DR, backup_t Data).....65
Web site.....7

History

Date	Modification
October2008	Initial version
January 2009	Update after development
February 2009	Revision before publishing
August 2009	Revision after v3.8 Circle OS release
September 2010	Revision after Open4 added, and v4.1 CircleOS release
October 2010	Specific addresses correction
January 2012	Revision after STM32L and STM3240G daugtherboard added
July 2013	Added STM32G memory map Added CX functions Added file system paragraph Added utilities paragraph Backup registers reorganization (fix volume format)
September 2013	Added STM3242x memory map Added DMA2D functions Changed FAT structure Fixed RAM addresses
November 2013	Fixed new FAT addresses Added SPI example for Cx management