UM1819
User manual

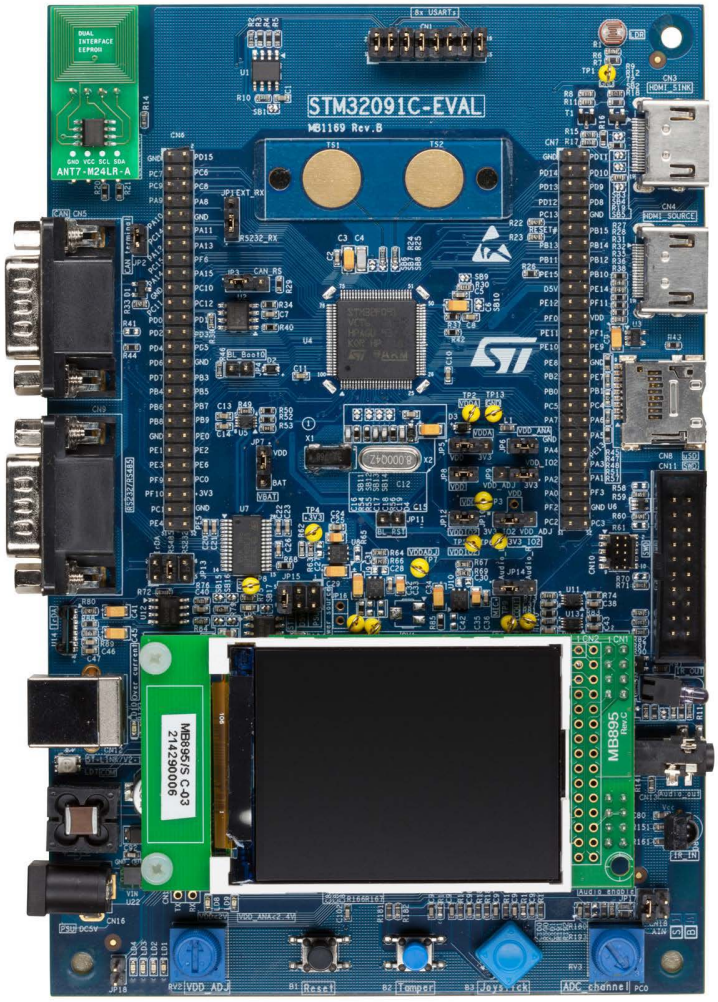
Demonstration firmware for STM32091C-EVAL board

Introduction

The STM32CubeF0 Demonstration comes on top of the STM32Cube™ as a firmware package that offers a full set of software components based on a modules architecture allowing re-using them separately in standalone applications. All these modules are managed by the STM32Cube™ Demonstration kernel allowing to dynamically adding new modules and access to common resources (storage, graphical and components).

The STM32CubeF0 Demonstration is built on the light kernel and on services provided by BSP, components, based on the STM32Cube™ HAL, using almost the whole STM32 capability to offer a large scope of usages.

Figure 1. STM32091C-Evaluation board



Contents

List of tables

List of figures

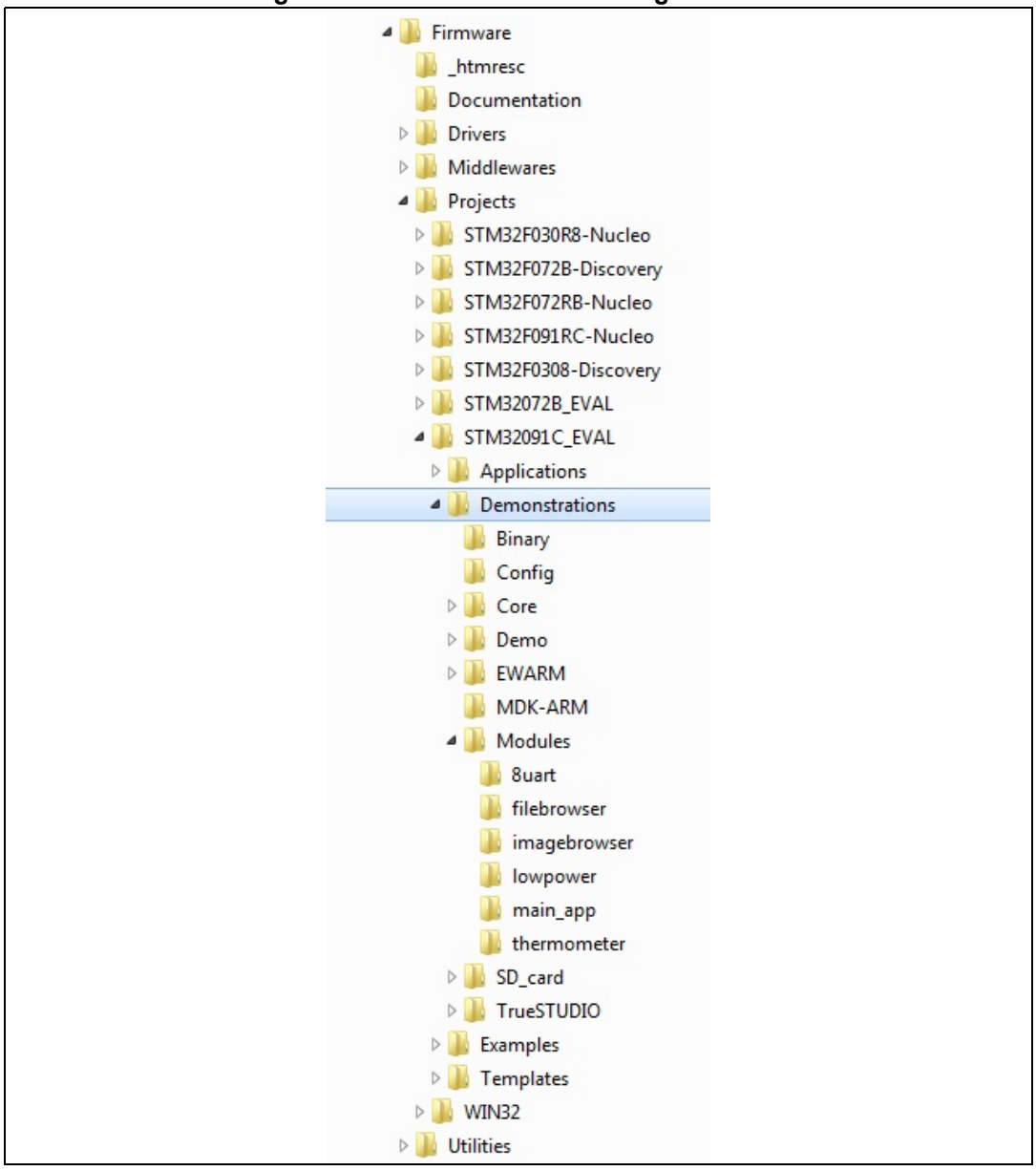
1 Demonstration description

The demonstration has been designed with three main objectives:

- Toolkit with low memory consumption.
- Modular applications: independents with high level of reuse.
- Basic menu navigation through joystick.

1.1 Demonstration folder organization

Figure 2. Demonstration folder organization

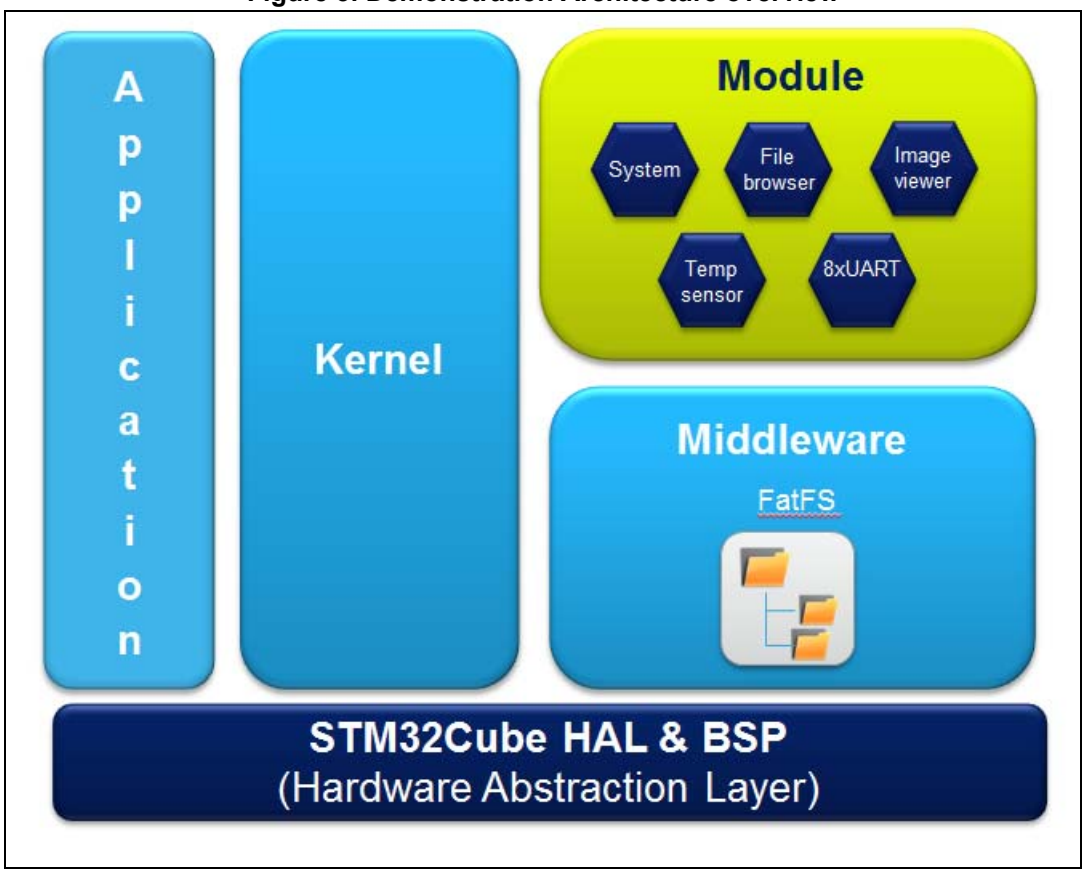


The demonstration sources are located in the “Projects” folder of the STM32Cube package for each supported board, here in STM32091C_EVAL folder. The sources are divided into five groups described as follows:

- **Binary**: demonstration binary file in Hex format
- **Config**: all middleware’s components and HAL configuration files
- **Core**: contains the kernel files
- **Demo**: contains startup files
- **Modules**: contains user modules including the graphical aspect and the modules functionalities.
- **SD_card**: resource files required by the demo that should be present in SD card
- **Project settings**: a folder per tool chain containing the project settings and the linker files.

1.2 Demonstration Architecture overview

Figure 3. Demonstration Architecture overview



The STM32CubeF0 Demonstration is composed of:

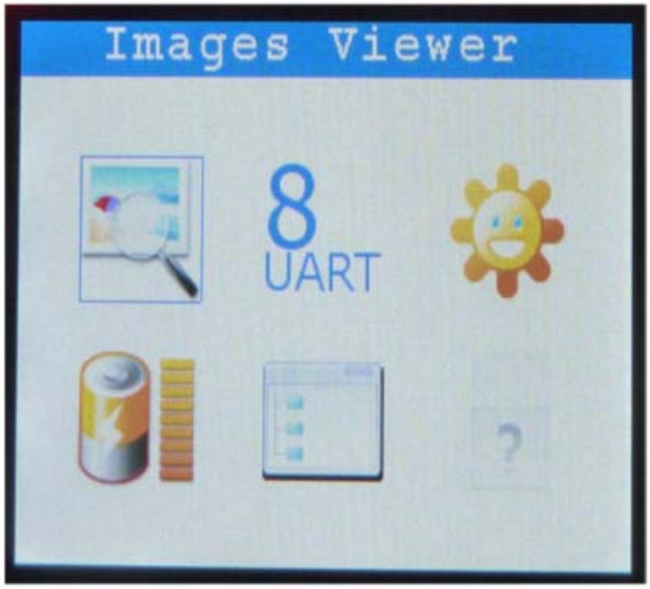
- **Application**: in charge to initialize kernel, HAL, interrupt handler and launch the main module
- **Kernel**: module scheduler and services provider (popup, files system, …)
- **Module**: graphical aspect and application execution
- **Middleware**: Only FATFS used
- **STM32Cube HAL & BSP**: provide access to HW and external component

2 Demonstration functional description

2.1 Menu

The purpose of the demonstration is to bring out the capabilities of microcontroller and Evaluation board peripherals. It runs only on the STM32091C_EVAL board. The demonstration contains the following applications. (Refer to *Figure 4*)

Figure 4. Demonstration main menu

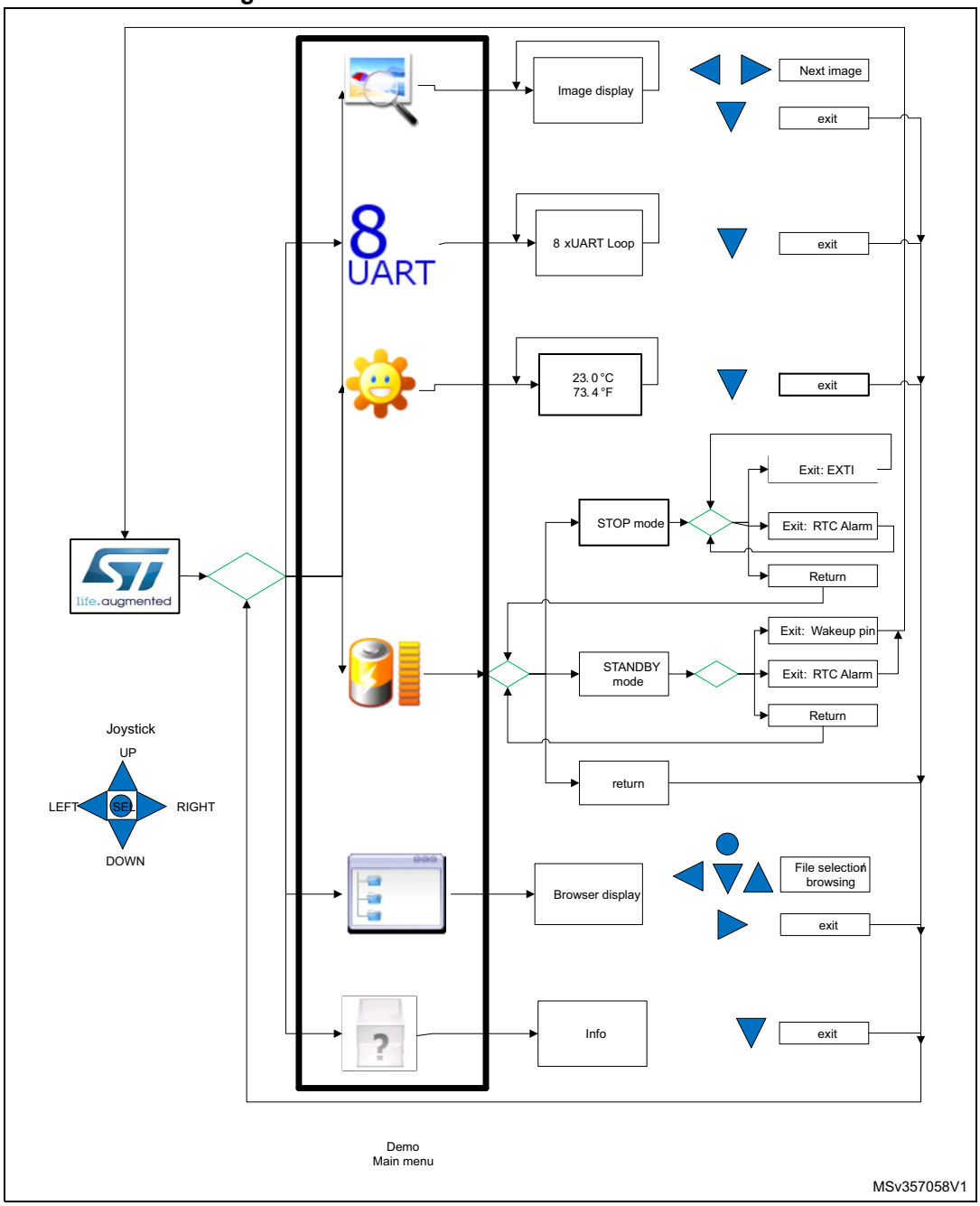


The UP, DOWN, RIGHT and LEFT joystick directions allow the user to navigate between items in the main menu and the submenus.

To enter a submenu, press the SEL push-button.

The SEL push-button designates the action of vertically pressing the top of the joystick, as opposed to moving it horizontally UP, DOWN, RIGHT or LEFT.

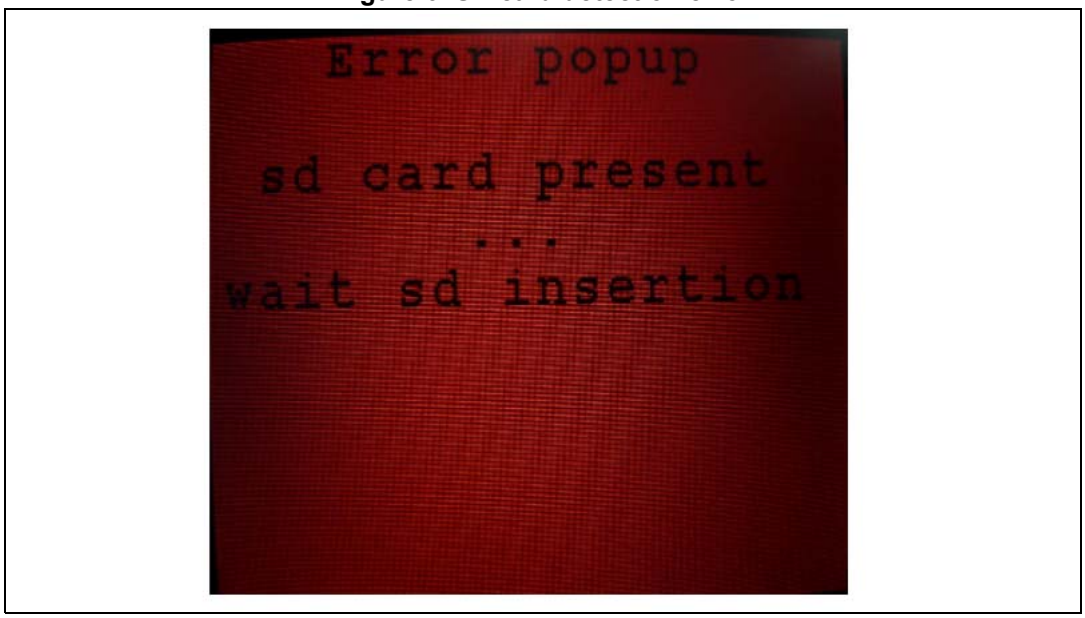
Figure 5. Structure of the demonstration menus



2.2 Demo startup

After a board reset, at demo startup, the system checks if a SD card memory is present in connector CN8. If no card is detected, the demo does not start and the message shown in *Figure 6* is displayed on the LCD screen.

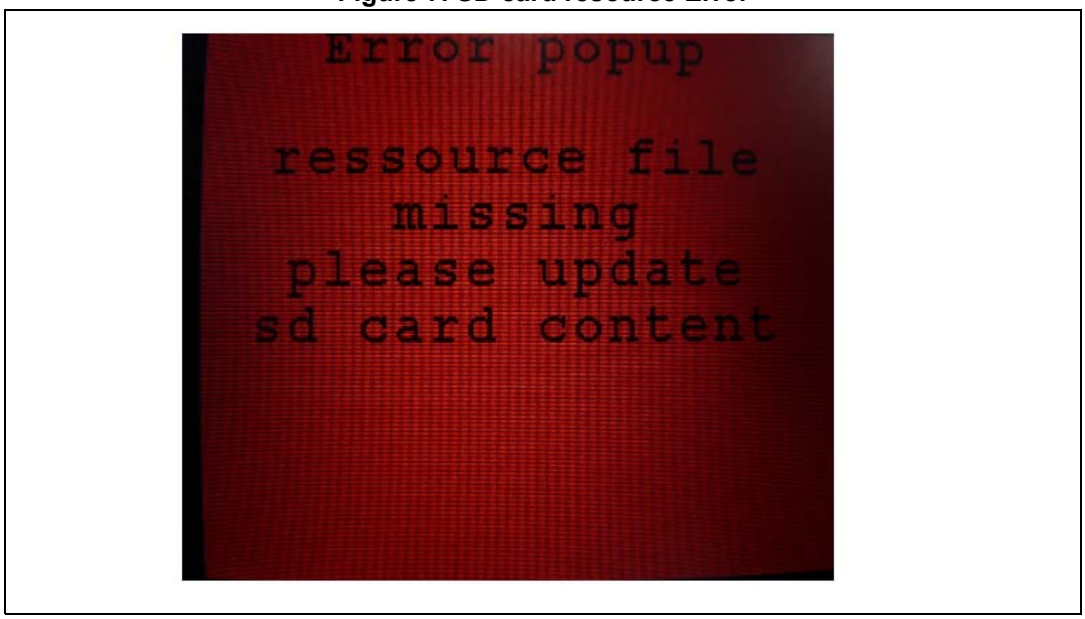
Figure 6. SD card detection error



The demo continues only if an SD card is inserted.

Then, each module will check if all needed graphic icons & bitmap files are present on SD card. If some resources are missing, the below message appears on the screen, and the demo does not continue.

Figure 7. SD card resource Error

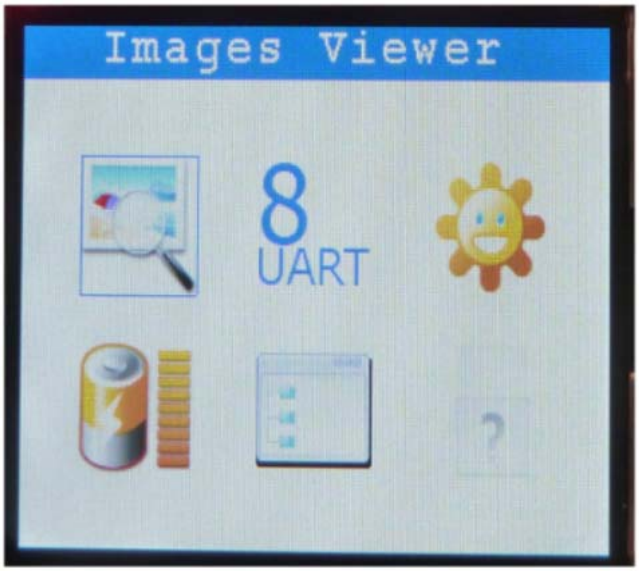


If all needed files are well present in SD Card memory, STM32091C_EVAL welcome screen is displayed and ST logo appears on LCD (see *Figure 8*).

Figure 8. ST Logo

Finally, the main menu is displayed as shown in *Figure 9*, and user can start enjoying applications.

Figure 9. Demonstration main menu



Note: Some icons shown in Figure 9 are taken from wikimedia

Once a submenu has been pre-selected, the associated icon is highlighted in blue color, the name of the module is displayed at the top of the screen.

2.3 Navigation

The demonstration menu is based on circular navigation, submenu selection, item selection and back navigation.

User navigates using joystick push-button located on evaluation board: RIGHT, LEFT, SEL, UP and DOWN.

- UP, DOWN, RIGHT and LEFT push-buttons are used to perform circular navigation in main menu and current menu items.
- SEL push-button selects the current item.
- UP and DOWN push-buttons are used for vertical navigation in submenus.

2.4 Modules

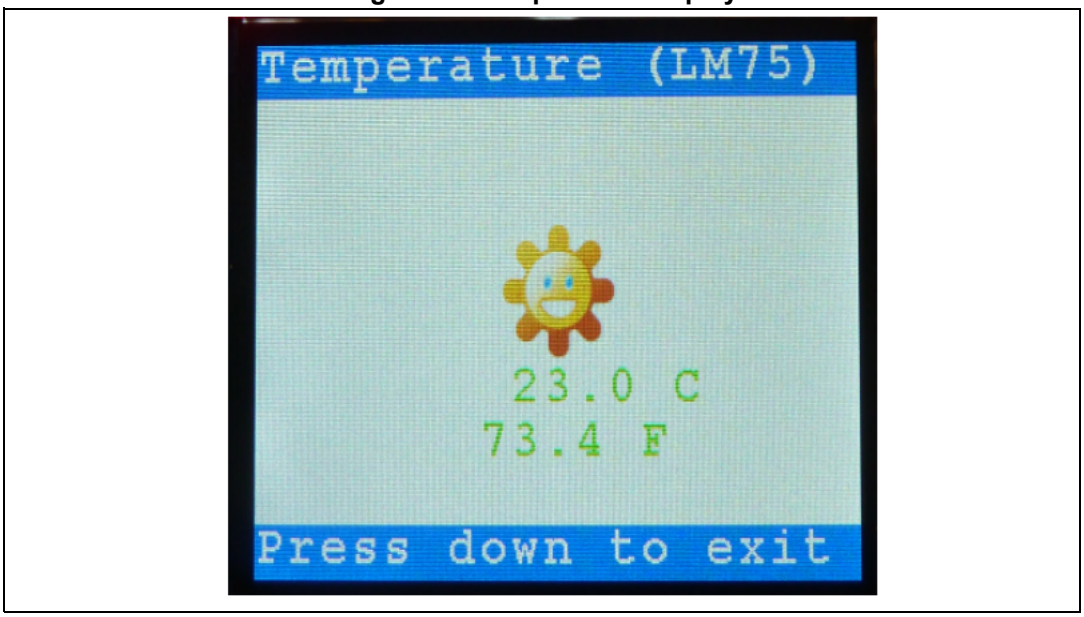
The following section provides a detailed description of each module of demonstration.

2.4.1 Thermometer

Overview

The STM32F091VCT6 microcontroller has two embedded I2C peripherals that can be connected to any device supporting the I2C protocol. A STLM75 (or a compatible device) I2C temperature sensor is mounted on the STM32091C-EVAL board and used to capture the external temperature (-55°C to +125°C). Once Temperature submenu has been selected by pressing SEL push-button, temperature value is displayed in Celsius and Fahrenheit as shown in *Figure 10*.

Figure 10. Temperature display



Press DOWN key to return to the main menu.

2.4.2 8xUART

Overview

This module intends to demonstrate the new capability of the STM32F091VCT6, to ensure Data buffer transmission and reception over 8 UARTs.

Hardware configuration

STM32091C_EVAL board is equipped with a new CN1 header for all UART Tx/Rx.

Before running this application, user shall ensure that all UARTs are chained together thanks to jumpers plugged on dedicated connector CN1.

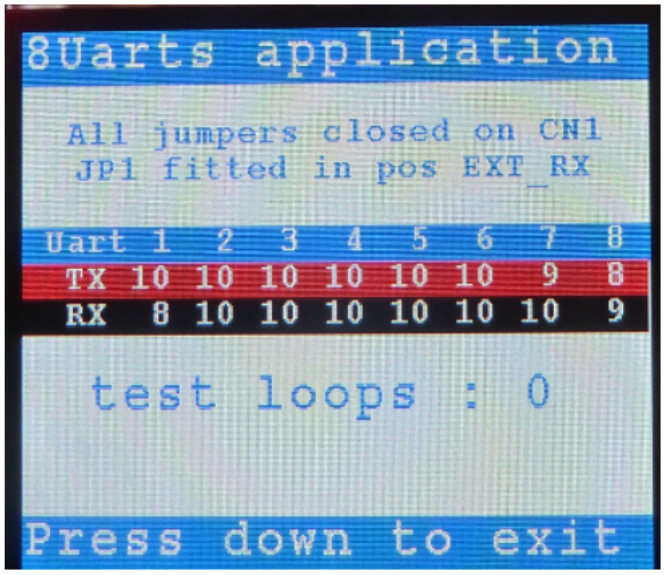
Also, jumper JP1 should be fitted in position 2<->3 in order to route (UART1Rx) PA10 to CN1.

Functional description

In this module, a transmit Buffer is predefined with a 180 chars string, that will be transmitted/received in loop by packet of 18 data through all 8 UARTs.

User will be able to monitor the number of packets transmitted/received on each UART, and the number of full transfer loops achieved.

Figure 11. 8xUARTs loopback



Press DOWN key to return to the main menu.

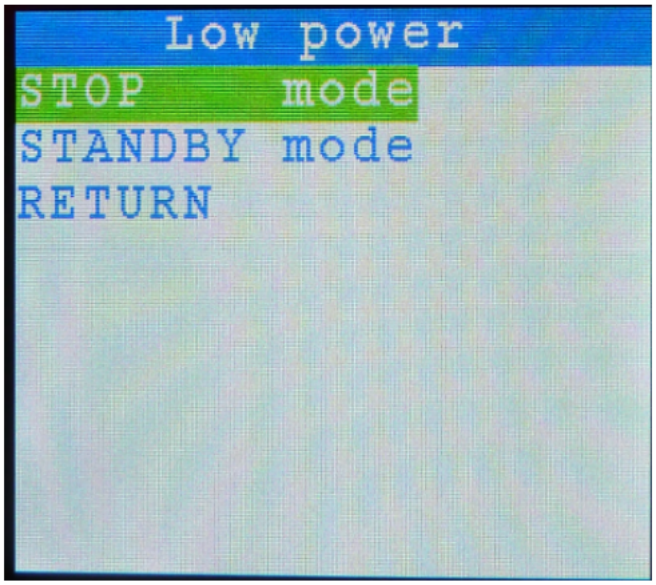
Note: In case one of the jumpers on CN1 is removed during the loopback test, the packet transmission/reception will be stopped.

2.4.3 Low power organization

Overview

STM32F091VCT6 microcontroller provides different operating modes in which the power consumption is reduced. The purpose of this menu is to show the behavior of microcontroller in different low-power modes. The Stop and Standby modes are taken as examples.

Figure 12. Low power menu



Functional description:

STOP mode

This menu allows user to put STM32F091VCT6 in STOP mode with Low power regulator ON and wake it up in two different ways:

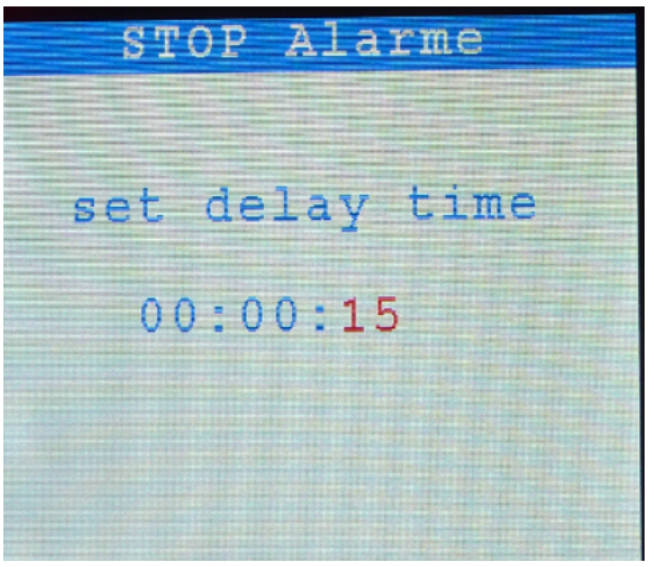
1. Tamper button configured as External Interrupt (WFI)
   – Press JOY_SEL to enter STOP mode. When the MCU is in Stop mode, the message shown in *Figure 13* is displayed on the LCD.

Figure 13. Stop mode entered with EXTI

MCU remains in Stop mode until Tamper push-button is pressed. System clock is then set to 48 MHz and the application resumes execution.

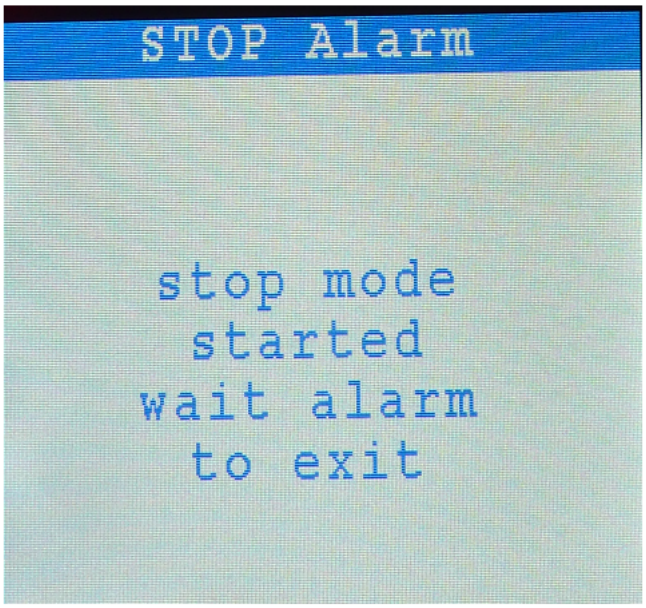
2. RTC Alarm (WFE)
   – When selecting this submenu, user has to set alarm delay when MCU has to exit from Stop mode. *Figure 14* shows how to set the wakeup time, using JOY_UP, JOY_DOWN, JOY_LEFT, JOY_RIGHT.

Figure 14. Setting wakeup time



   – Wakeup time is validated after a press on JOY_SEL, and MCU enters then in STOP mode.

Figure 15. Stop mode entered, wait RTC alarm



RTC Alarm wakes up MCU from Stop mode after programmed time has elapsed.

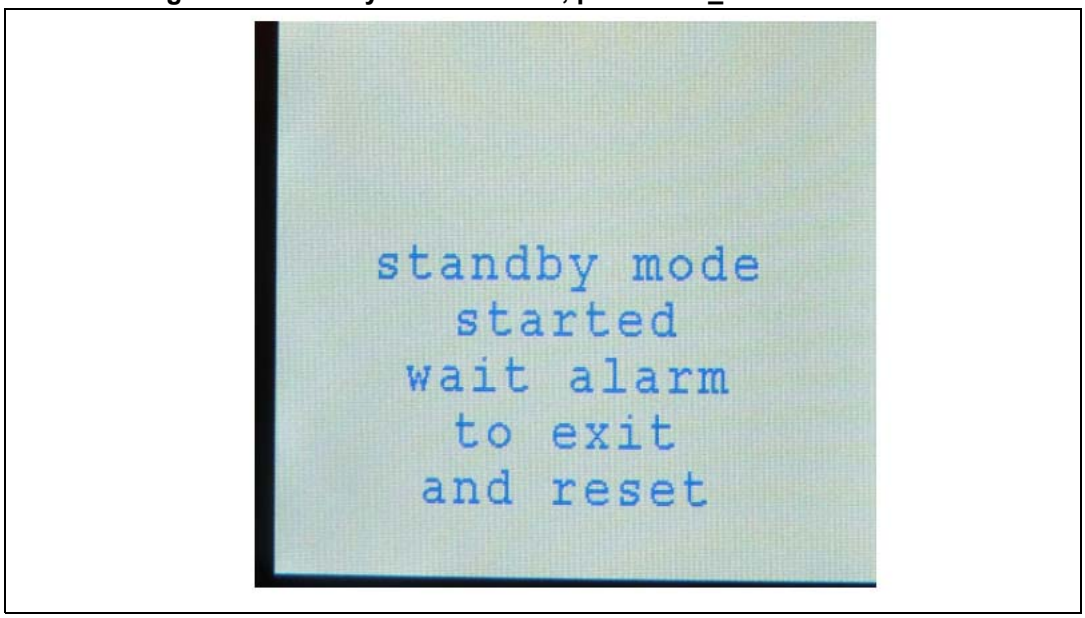
The system clock is then restored to 48 MHz and application resumes execution.

STANDBY mode

This menu allows user to put STM32F091VCT6 in Standby mode, and wake it up in two different ways:

1. JOY_SEL button configured as Wake-up pin
   – Wakeup push-button is used to wake up the MCU from the Standby mode. Once the Standby mode> EXIT Wakeup submenu has been selected, user shall then press SEL push-button to allow the system entering Standby mode.

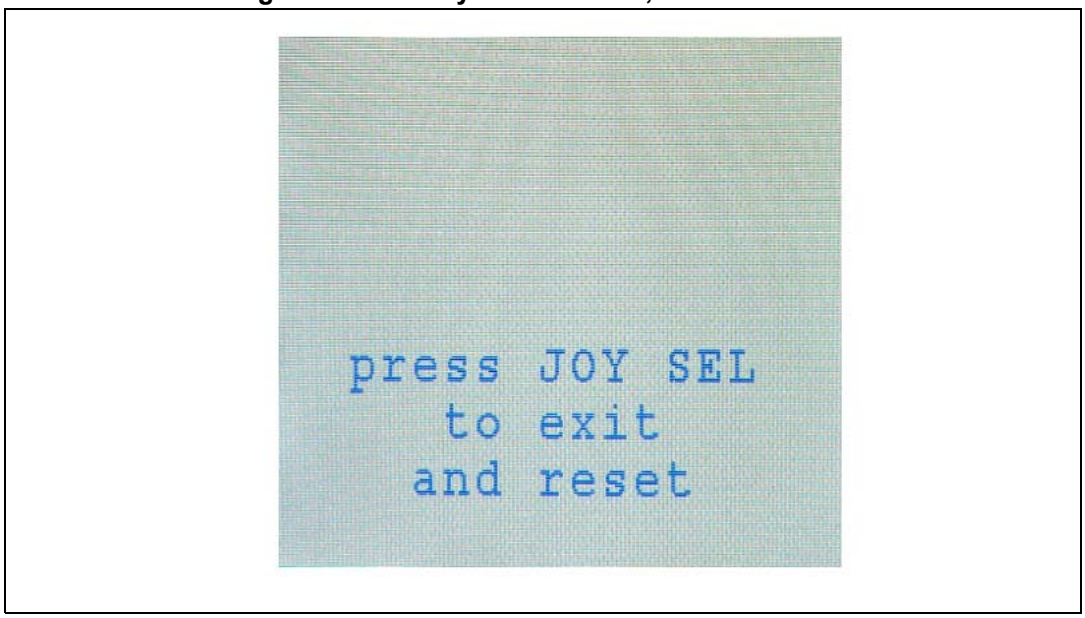
Figure 16. Standby mode entered, press JOY_sel to exit & reset



MCU remains in Standby mode until Wakeup push-button is pressed. Once the Wakeup push-button has been pressed, MCU exits Standby mode and program execution resets.

2. RTC Alarm
   - When selecting this submenu, user has to set alarm delay when MCU has to exit from Standby mode.
   - RTC Alarm wakes up the MCU from the Standby mode after the programmed time has elapsed. When selecting this submenu, the user has to set the alarm to the time when the MCU has to exit from Standby mode.
   - User can set the wakeup time, using JOY_UP, JOY_DOWN, JOY_LEFT, JOY_RIGHT. Once Wakeup time is validated with a press on JOY_SEL, MCU enters then in STANDBY mode.

Figure 17. Standby mode started, wait RTC alarm



After the programmed timing has elapsed, the system exits the Standby mode and program execution resets.

2.4.4 About

Overview

This submenu shows version & date of STM32091C_EVAL demo firmware. When“About menu” is selected, the message shown in *Figure 18* is displayed on the LCD screen.

Figure 18. About menu

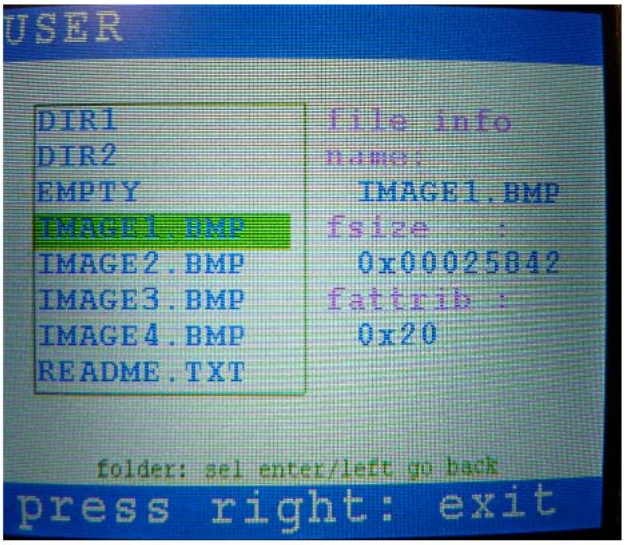
Press DOWN key to return to the main menu.

2.4.5 File Browser

Overview

File browser module is a system module that allows exploring the folder “USER” from connected storage unit(s) – here SD card. It allows user to display file information and browse sub-directories. The file list structure is built during the media connection.

Figure 19. File browser display



Functional description

File browser is light and allows basic file system operations like: explore folder, file information. Others operations like file deletion or opening supported extension file are not supported.

Pressing UP/DOWN joystick allows user to browse files and sub- directories. When sub-directory is selected (highlighted in green), user can enter it by pressing joy selector. Pressing joystick LEFT, allows to go up in File System structure or go back to main menu, if already in root directory.

2.4.6 Image viewer

Overview

Image viewer module allows to retrieve BMP images stored on SD card & to display them in full screen on the LCD.

When Image viewer is selected, a first image is displayed on the screen. User can then use RIGHT or LEFT push-button to view next/previous image stored on SD card.

This application reads all bitmap pictures from USER directory (see *Section 4.4: Programming the demonstration*) and displays only the BMP files having the following format:

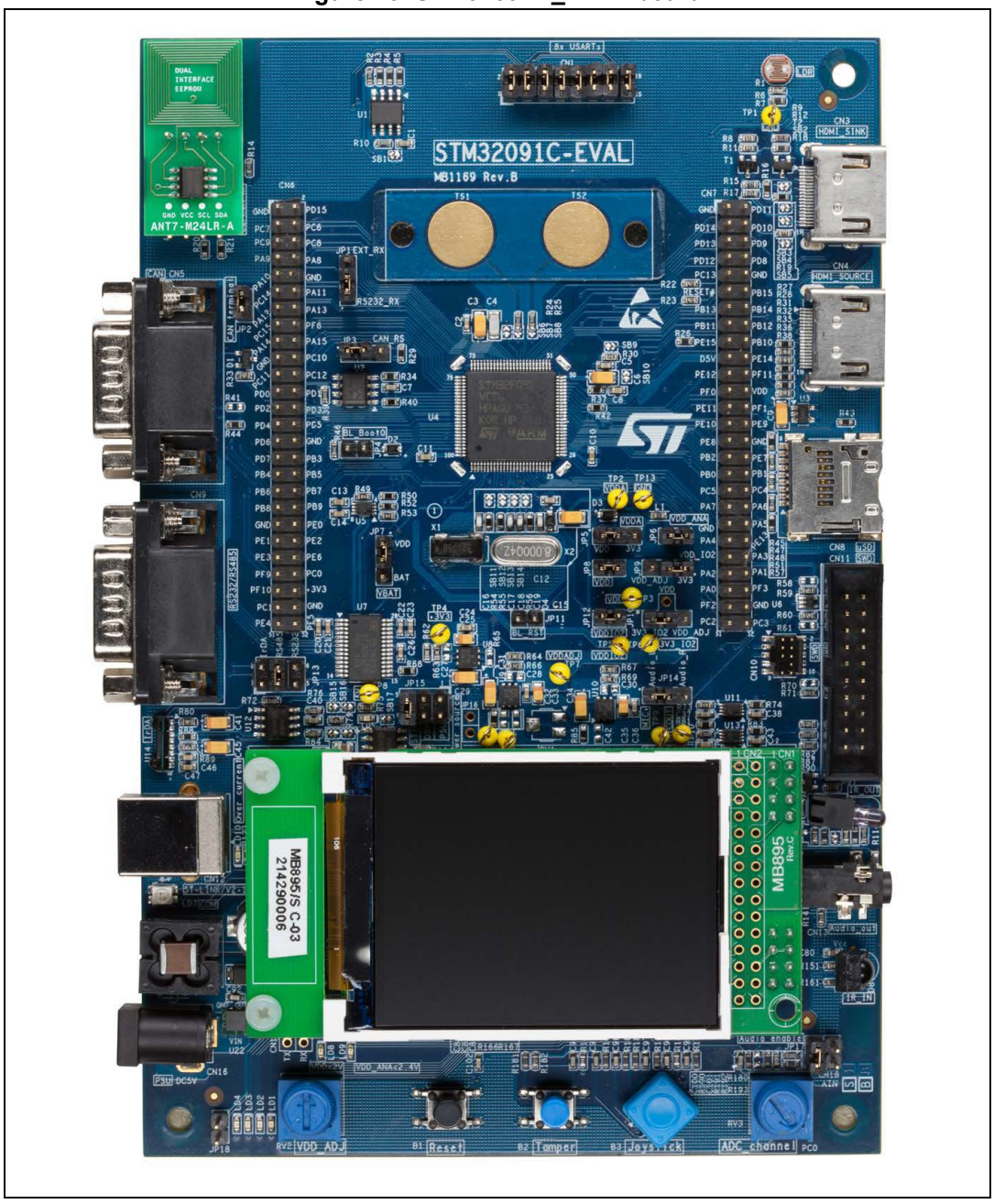
- Bit depth: 16 bits (RGB)
- Size: 240x320

Press DOWN or UP push-button to return to main menu.

3 Hardware settings

The STM32Cube demonstration supports STM32F091xC device and runs on STM32091C_EVAL board from STMicroelectronics.

Figure 20. STM32091C_EVAL board



For the STM32091C-EVAL board the following default jumper's configuration is used.

Table 1. Default jumper's configuration

Jumper	Position description
JP18	Not fitted
JP17	fitted: Speaker amplifier enabled
JP15	Only STlk is fitted to power the board from USB or ST-LINK/V2-1
JP14	<1-2>: Mono playback enabled
JP13	RS232 fitted: RS232_RX is connected to RS232 transceiver and RS232 communication is enabled
JP12	fitted
JP11	Not fitted
JP10	<1-2>: VDDIO2 connected to 3.3V
JP9	<2-3>
JP8	fitted
JP7	<1-2>: VBat pin connected to VDD
JP6	fitted
JP5	<1-2>
JP4	Not fitted
JP3	<1-2>: CAN receiver working in high-speed mode
JP2	Not fitted
JP1	<2-3> (used for USARTx look-back test)
CN1	All jumpers should be fitted

4 Software settings

4.1 Clock Control

STM32F091VCT6 internal clocks are derived from the HSE (clocked by the external 8 MHz crystal).

In this demo application, the various system clocks are configured as follows:

- System clock is set to 48 MHz: the PLL is used as the system clock source.
- HCLK frequency is set to 48 MHz.

Only the RTC is clocked by a 32 kHz external oscillator.

4.2 Peripherals

All used peripherals are described in *Table 2*

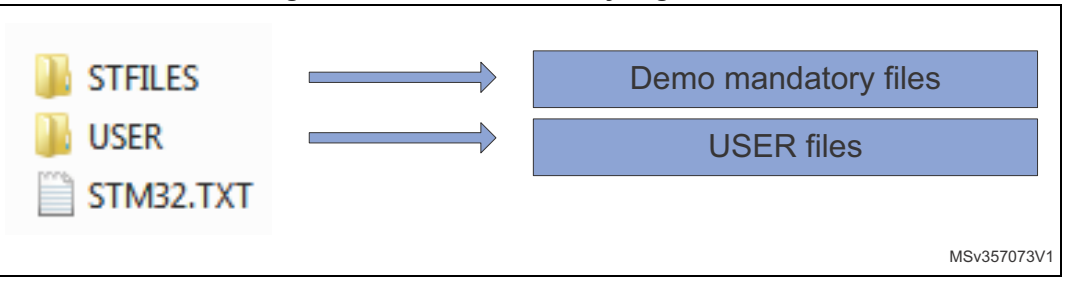
Table 2. Peripherals

Used peripherals	Application/Module
I2C1	Temperature sensor
EXTI	Menu navigation + joystick + push-button + low power mode + audio + Applications
GPIO	All Applications
PWR	Low power modes application
RCC	All Applications + Demo kernel
RTC	Low power mode application
SPI	Color LCD + SD card
UART	8xUART application

4.3 Media files

STM32091C_EVAL board comes with a MicroSD card memory preprogrammed with Image resources, txt files and directories tree used by demonstration FW. However, you can load your own image (*.bmp) files in the USER directory, assuming that file formats are supported by the demonstration.

Figure 21. SD Card directory organization



4.4 Programming the demonstration

User can program the demonstration using two methods:

4.4.1 Using Binary file

To program demonstration's binary image into the internal Flash memory, you have to use the STM32CubeF0_Demo_STM32091C_EVAL.hex file located under Project\STM32091C-EVAL\Binary.

4.4.2 Using preconfigured projects

Select the folder corresponding to your preferred toolchain (MDK-ARM, EWARM or TrueSTUDIO).

- Open STM32F091C-EVAL_Demo project and rebuild all sources.
- Load the project image through your debugger.
- Restart the evaluation board (press B1: reset button).

5 Software description

Following sections will describe in details:

- Application: in charge to initialize kernel, HAL, interrupt handler and launch the main module
- Kernel: module scheduler and services provider (popup, files system, …)
- Module: graphical aspect and application execution
- Middleware: Only FATFS used

5.1 Application

The application goal is to prepare demonstration startup, by initializing all the HW/SW before the main module execution (first action done by the kernel is to run a default module ID, please refer to chapter k_demo for more detail). Table below provides a description of all the actions performed by the different functions.

Table 3. Function Description

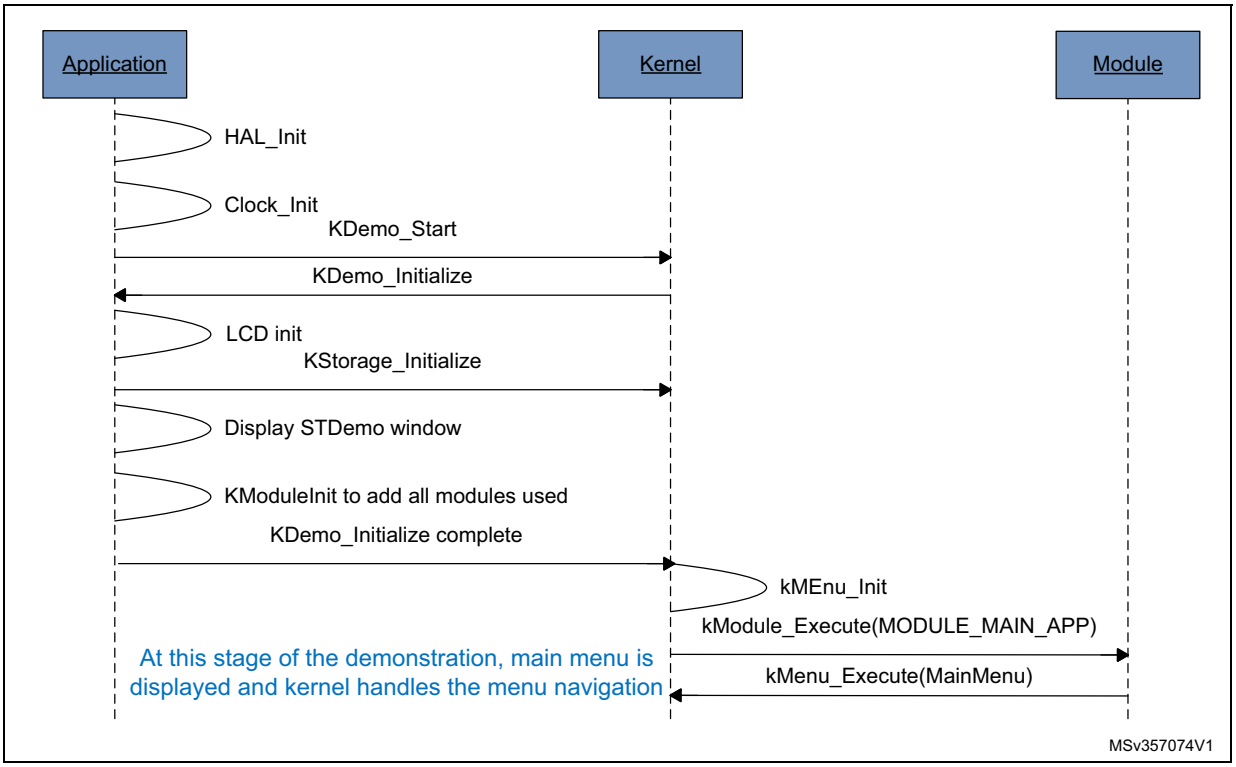
Functions (Main.c)	Descriptions
Main	initialize the HAL, configure the clock and start the demonstration
kModule_Init	Initialize the kModule with all the modules
kDemo_Initialization	Initialize the LCD, check SD state, initialize FatFs, display the startup window, initialize the module and check the modules resources
kDemo_UnInitialization	Implemented however performs no action
HAL_GPIO_EXTI_Callback	According the PIN id forward SD detection or Joystick event to kernel

The file “stm32f0xx_it.c” is also part of the application and is used, as usual, to map the interrupt vector on the driver HAL driver, depending on the module requirement.

5.2 Application startup overview

The following graph summarizes the application startup until the kernel takes the lead.

Figure 22. Application startup overview



5.3 Kernel

5.3.1 Overview

The goal of demonstration kernel is mainly to provide a generic platform that controls and monitors all the modules with minimum memory consumption. The kernel provides also a set of friendly services that simplifies module's implementation. Table below provides files kernel list with a short description.

Table 4. Files kernel list

Kernel	Description
K_demo	Initialize the demo
K_menu	Handle the menu navigation and functionality execution
K_module	Initialize the module
K_storage	Provide a set of function around the storage
K_tools	Provide a set of function tools
K_window	Provide a set of function to display a popup window

5.3.2 K_demo

The function “Kdemo_Start” is executed by the application to launch modules scheduling and the kernel starts by running the module with “ID=MODULE_MAIN_APP”. To keep the kernel independent with the HW/SW, functions KDemo_initialization and KDemo_UnInitialization are defined on application side and called by kDemo_Start (see application chapter for details about these functions)

Table 5. K_demo_Start function description

Function	Description
KDemo_Initialization	Initialize the demo (function defined on application side)
kDemo_UnInitialization	Initialize the demo (function defined on application side)
kDemo_Start	Start the demo (Initialize, Run, UnInitialize, Exit)
kDemo_Execute	Initialize kMenu and execute the module “MODULE_MAIN_APP”

5.3.3 K_menu

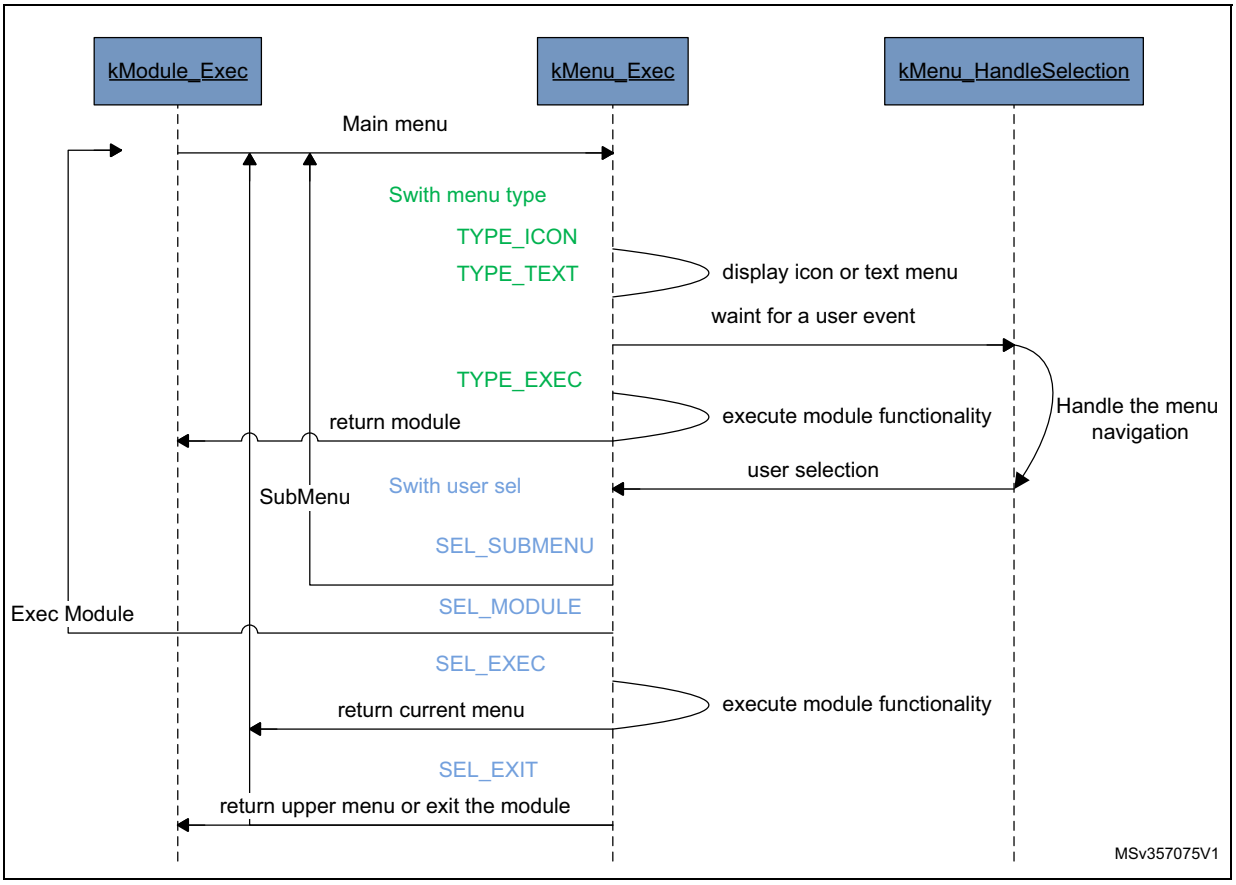
The module execution is started by a call to the function kMenu_Execute. This kernel function handles the menu navigation and the execution functionalities (thanks to the structure t_menu defined inside the module).

Table 6. K_menu function description

Function	Description
kMenu_Init	Initialize the joystick
kMenu_EventHandler	GPIO Event handler
KMenu_Execute	Function to execute a menu
kMenu_Header	Function to display header information
kMenu_HandleSelectionExecute	The function handle the menu navigation

The Graph below shows the execution flow for a menu:

Figure 23. K_menu execution flow



5.3.4 K_module

Table 7. K_module function description

Function	Description
kModule_Init	Defined on application side
kModule_Add	Add module inside the module list
kModule_CheckRessource	Check the module resource
kModule_Execute	Execute the module

This kernel part centralizes information about all modules available inside demonstration fw; Function kModule_Init (defined on application side) aims to register all the modules present whit the help of function kModule_Init (thanks to the structure K_ModuleItem_Typedef defined inside the module).

5.3.5 K_Storage

The K_Storage handles only the SD card storage and provides some services to simplify the modules development.

Table 8. K_Storage function Description

Function	Description
kStorage_Init	Mount the SD card file system
kStorage_DeInit	Unmount the file system
kStorage_GetStatus	Return SD card presence
kStorage_GetDirectoryFiles	Return the name of the file present inside a directory
kStorage_FileExist	Check if a file exist
kStorage_GetFileInfo	Return file information
kStorage_OpenFileDrawBMP	Open a bmp file and display it (only file of 8ko max)
kStorage_OpenFileDrawPixel	Open a bmp file and display it line by line
kStorage_GetExt	Return the file extension

5.3.6 K_tools

The K_Tools provides tools for module development.

Table 9. K_tools function description

Function	Description
kTools_Buffercmp	Return 0 if the both buffers are identical

5.3.7 K_Window

The k_Window provides services to display popup window without user event management (must be handled outside, for example inside the module).

Table 10. K_Window function description

Function	Description
kWindow_Popup	Display a popup
kWindow_Error	Display a red popup with an error message

5.3.8 Memory management

Nothing specific to optimize the memory management, only two recommendations:

- kernel is recursive, so the stack allocation must be aligned with the deep of the modules chains.
- Each module allocates memory, for optimization purpose the allocation must be done on the heap (kModulePreExec allocate and kModulePreExec release the memory)

The demonstration is currently using:

– CSTACK = 0x1800
– HEAP = 0x3000

5.4 Module

A module is an autonomous application that can run directly from the launcher or from another module. The module contains two main parts:

- Module control: the kernel used this part to handle the display and interact with the end-user.
- Functionalities: execution function(s)

5.5 Module control

5.5.1 K_ModuleItem_Typedef

This structure is equivalent to a main function for the module:

Table 11. K_ModuleItem_Typedef

File	code[Byte]
kModuleId	Unique ID module (defined inside k_config.h)
kModulePreExec	the function is used to allocate memory or to execute any specific action before the module execution (this function is optional)
kModuleExec	Entry point used to start the main menu of the module
kModulePostExec	The function is used to release the resource allocation done inside kModulePreExec (this function is optional)
kModuleRessouceCheck	Function used to control if the resource of the module are available (this function is optional)

5.5.2 kModulePreExec & kModulePostExec

Both functions are optional and used mainly for memory optimization (memory allocated only when the module is running)

Figure 24. kModulePreExec example:

```
/**
  * @brief  setup the HW/Memory for the 8 uart application
  * @param  None.
  * @note   set the memeory + Hw initialisation.
  * @retval None.
  */
KMODULE_RETURN _HuitUartConfig(void)
{
  uint8_t index;

  /* Memory allocation for the example */
  for(index = 0; index < USART__INDEX_MAX; index++)
  {
      aRxBuffer[index] = malloc(sizeof(uint8_t)*BUFFER_SIZE);
```

Figure 25. kModulePostExec example:

```
/**
  * @brief  unsetup the HW for the 8 uart application
  * @param  None.
  * @note   reset the memeory + Hw initialisation.
  * @retval None.
  */
KMODULE_RETURN _HuitUartUnConfig(void)
{
  uint8_t index;

  /* free the memory allocated */
  /* Memory allocation for the example */
  for(index = 0; index < USART__INDEX_MAX; index++)
  {
      free(aRxBuffer[index]);
```

5.5.3 kModuleExec

This function is mainly used to execute the module and start the menu execution of the modules.

Figure 26. kModule example:

```
/**
  * @brief  Run the 8 uart application
  * @param  None.
  * @note   run and display information about the uart transaction.
  * @retval None.
  */
KMODULE_RETURN _HuitUartExec(void)
{
  /* Prepare the main MMI */
  BSP_LCD_Clear(LCD_COLOR_WHITE);
  kMenu_Execute(HuitUartMenu);

  /* Execute the app 8uart */
  /* App initialization    */
  return KMODULE_OK;
}
```

5.5.4 kModuleRessourceCheck

The function is used to control if resources are available: for example control if bmp file are well present on the SD card.

Figure 27. kModuleRessourceCheck

```
/**
  * @brief  check the main application ressources
  * @param  None.
  * @note   None.
  * @retval None.
  */
KMODULE_RETURN AppMainExecCheckRessource()
{
 uint8_t index;

  /* check icon menu */
  for(index = 0; index <  countof(MainMenuItems); index++)
  {
    if(kStorage_FileExist(MainMenuItems[index].pIconPath) != KSTORAGE_NOERROR)
    {
      return KMODULE_ERROR_ICON;
    }
  }
  return KMODULE_OK;
}
```

5.6 Graphical aspect

The graphical aspect consists in describing, inside structure, the menu architecture.

5.6.1 The structure tMenu

This structure is the main entry point used by function kMenuExecute, to display the module MMI and execute the functionalities.

Table 12. tMenu structure

File	code[Byte]
pszTitle	Menu title
psItems	Pointer on the menu data
nItems	Number of entry inside menu data pointer
nType	Menu Type:
	TYPE_ICON, TYPE_TEXT, TYPE_EXEC
line	In case of icon, number of icon line
column	In case of icon, number of icon column

5.6.2 The structure tMenuItem

This structure is the menu item description

Table 13. tMenuItem structure

File	code[Byte]
pszTitle	Menu title
x, y	Icon position on the screen
SelType	Type de selection
	SEL_EXEC: execute a functionality
	SEL_SUBMENU: select a sub menu
	SEL_EXIT: exit the current menu and return to previous menu or module
	SEL_MODULE: execute a module
ModuleId	Module ID
pfExecFunc	Function to execute
pfActionFunc	Callback used to capture the user feedback
psSubMenu	Pointer on a submenu

5.6.3 Menu Architecture

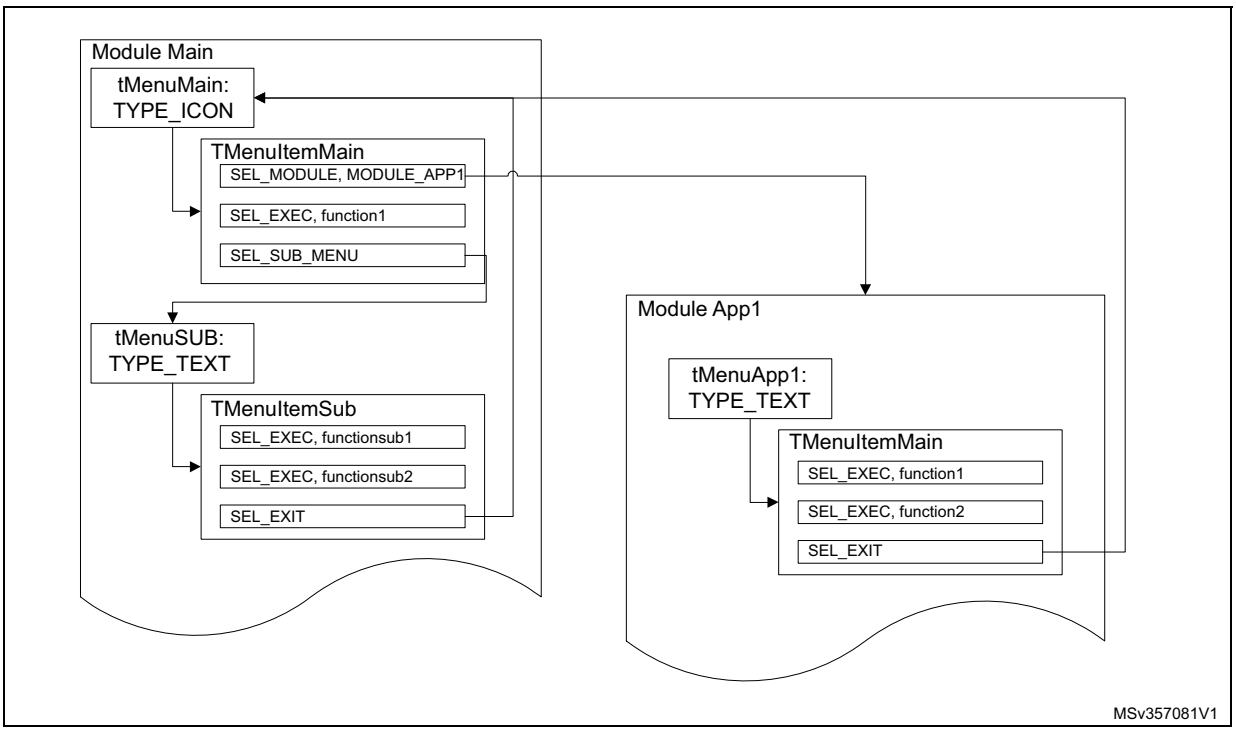
The figure below shows an example of a basic menu structure with the use of two modules:

- The module “Main” is built with two levels of menu and three functionalities. The level main is an “ICON” menu which can execute another module, execute the function1 or

display a sub “TEXT”. This submenu can execute functionsub1, functionsub2 or go back on “Main” Menu.

- The module “App1” is a “TEXT” with 2 embedded functionalities function1, function2.

Figure 28. menu architecture inside a module



5.7 Functionality

Functionality is an action selected by the end-user through the menu. This menu event will call of a function which must respect the following prototype:

```
void functionExec(void)
```

On the function exit, the kernel will return into the previous menu state.

In many cases, the functionality will be stopped by an user event. So the kernel offers the capability to return all the events through a callback function with the following prototype:

```
void functionSel(uint8_t sel)
```

The functionality settings are done inside tMenuItem through the fields: pfExecFunc and pfActionFunc.

5.8 Adding a new module

Once the module appearance and functionality are defined and created, based on constraints described above, only the module is left to be added:

1. Define unique ID of the new module in k_config.h file.
2. k_ModuleAdd() function should be called in KDemo_Initialization (), with unique ID defined in step1.
3. Modify main module to add the call to this new module (See *Figure 28*).

5.9 Middleware (FatFS)

Only one middleware has been integrated inside the demonstration toolkit: FatFS. The full APIs functions set given by the file system interface are:

Table 14. APIs functions description

Function	Description
f_mount	Register/Unregister a work area
f_open	Open/Create a file
f_close	Close a file
f_read	Read file
f_write	Write file
f_lseek	Move read/write pointer, Expand file size
f_truncate	Truncate file size
f_sync	Flush cached data
f_opendir	Open a directory
f_readdir	Read a directory item
f_getfree	Get free clusters
f_stat	Get file status
f_mkdir	Create a directory
f_unlink	Remove a file or directory
f_chmod	Change attribute
f_utime	Change timestamp
f_rename	Rename/Move a file or directory
f_mkfs	Create a file system on the drive
f_forward	Forward file data to the stream directly
f_chdir	Change current directory
f_chdrive	Change current drive
f_getcwd	Retrieve the current directory
f_gets	Read a string

Table 14. APIs functions description (continued)

Function	Description
f_putc	Write a character
f_puts	Write a string
f_printf	Write a formatted string

6 Footprint

This chapter sums up Ram / Rom consumption per software blocks. Here is full demonstration software consumption:

Table 15. Software consumption

Full demonstration software	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
	55150	18272	25500

6.1 Kernel footprint

The following table shows the code memory, data memory and the constant memory used for each kernel file:

Table 16. Kernel footprint consumption

File	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
k_demo	60	-	-
k_menu	1436	-	4
k_module	220	-	160
k_storage	884	52	1111
k_tools	38	-	-
k_window	252	12	-
main	682	204	-

Note: In this section, memory is sometimes allocated dynamically in some structures in order to save maximum RAM consumption.

6.2 Module footprint

The following table purpose shows the code memory, data memory and the constant memory for each module:

Table 17. Module footprint consumption

Module	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
8uart_app	3264	692	65
filesbrowser_app	1852	304	2330
imagesbrowser_app	240	84	2

Table 17. Module footprint consumption (continued)

Module	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
lowpower_app	1 624	824	34
main_app	308	552	1
thermometer_app	324	176	1

Note: In this section, memory is sometimes allocated dynamically in some structures in order to save maximum RAM consumption.

6.3 HAL footprint

The following table purpose shows the code memory, data memory and the constant memory for each HAL file:

Table 18. HAL footprint consumption

Module	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
stm32f0xx_hal	116	-	4
stm32f0xx_hal_cortex	304	-	-
stm32f0xx_hal_gpio	1152	-	-
stm32f0xx_hal_i2c	2208	-	-
stm32f0xx_hal_msp_template	2	-	-
stm32f0xx_hal_pwr	156	-	-
stm32f0xx_hal_rcc	396	16	-
stm32f0xx_hal_rcc_ex	2404	32	-
stm32f0xx_hal_rtc	1256	-	-
stm32f0xx_hal_spi	1996	-	-
stm32f0xx_hal_uart	2116	-	-
stm32f0xx_hal_uart_ex	356	-	-

6.4 BSP footprint

The following table purpose shows the code memory, data memory and the constant memory for each BSP file:

Table 19. BSP footprint consumption

Module	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
stm32091c_eval	2276	55	200
stm32091c_eval_lcd	1536	14836	920
stm32091c_eval_sd	1450	1	1
stm32091c_eval_tsensor	156	-	6

6.5 BSP components footprint

The following table purpose shows the code memory, data memory and the constant memory for each BSP component file:

Table 20. BSP components footprint consumption

Module	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
hx8347d	1228	28	697
spfd5408	1204	28	697
stlm75	224	8	16

6.6 Third party footprint

The following table purpose shows the code memory, data memory and the constant memory for each Third party (FatFS) file:

Table 21. Third party footprint consumption

Module	Read only code memory [Byte]	Read only data [Byte]	Read Write data memory [Byte]
diskio	220		
ff	7924	144	30
ff_gen_drv	84		12
sd_diskio	268	11	21

7 Revision history

Table 22. Document revision history

Date	Revision	Changes
24-Oct-2014	1	Initial release.