UM2145
User manual

STM32CubeL4 demonstration firmware for 32L496GDISCOVERY kit

Introduction

STMCube™ is an STMicroelectronics original initiative to make developers' lives easier by reducing development effort, time and cost. The STM32Cube covers the whole STM32 portfolio.

STM32Cube Version 1.x includes:

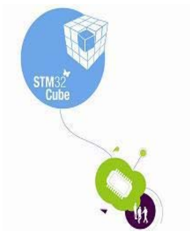
- The STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL4 for STM32L4 series)
  – The STM32CubeL4 HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio.
  – Low Layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  – A consistent set of middleware components such as RTOS, USB, FatFS, graphics.
  – All embedded software utilities delivered with a full set of examples.

The STM32CubeL4 32L496GDISCOVERY demonstration platform is built around the STM32Cube HAL, BSP and RTOS middleware components.

This evaluation board is suitable hardware to evaluate STM32L4 ultra-low-power solutions and audio/graphic capabilities thanks to a capacitive touchscreen LCD-glass display, two microphones, a joystick, external PSRAM and Quad-SPI Flash memories, an ST-LINK/V2 debugger/programmer and an STM32L496AG microcontroller.

The architecture was defined with the goal of making from the STM32CubeL4 32L496GDISCOVERY demonstration core an independent central component, which can be used with several RTOS and third party firmware libraries through several abstraction layers inserted between the STM32CubeL4 32L496GDISCOVERY demonstration core and the several modules and libraries working around it.

The STM32CubeL4 32L496GDISCOVERY demonstration supports STM32L496xx devices and runs on the 32L496GDISCOVERY board.



Contents

List of tables

List of figures

1 STM32CubeL4 main features

STM32CubeL4 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32L4 microcontrollers. In line with the STM32Cube initiative, this set of components is highly portable, not only within the STM32L4 series but also to other STM32 series.

STM32CubeL4 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in an open-source BSD license for user convenience.

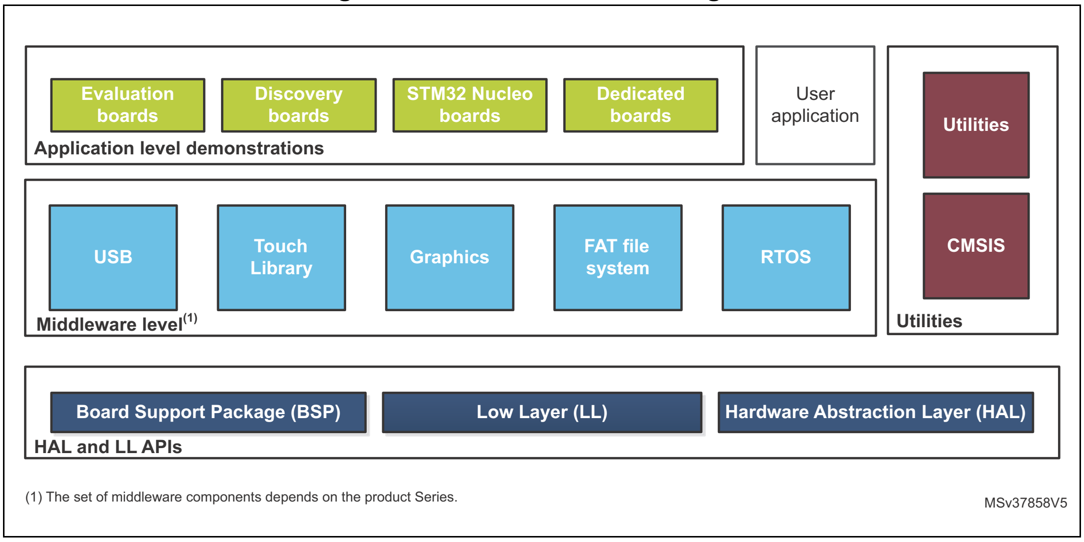
STM32CubeL4 package features a set of middleware components with the corresponding examples. They come with very permissive license terms:

- Full USB stack supporting many classes (HID, MSC, CDC, Audio, DFU)
- CMSIS-RTOS implementation with FreeRTOS open source solution
- FAT File system based on open source FatFs solution
- STMTouch touch sensing solution.

A demonstration implementing all these middleware components is also provided in the STM32CubeL4 package.

The block diagram of STM32Cube is shown in *Figure 1*.

Figure 1. STM32Cube block diagram



2 Getting started with the demonstration

2.1 Hardware requirements

The hardware requirements to start the demonstration application are as follows:

- 32L496GDISCOVERY board (see *Figure 2*) (refer to UM2160 for Discovery board description)
- One “USB type A to micro-B” cable to power up the STM32 Discovery board from the USB ST-LINK (micro-B USB connector CN5)

The 32L496GDISCOVERY board helps you to discover the ultra-low-power features and audio/graphic capabilities of the STM32 L4 series. It offers everything required for beginners and experienced users to get stared quickly and develop applications easily.

Based on an STM32L496AGT6 MCU, the 32L496GDISCOVERY board includes an ST-LINK/V2-1 embedded debug tool interface, an Idd current measurement panel, external PSRAM and QuadSPI flash, an audio codec with 3.5mm connector, a Capacitive Touch Panel LCD screen (240x240 pixels), LEDs, a joystick and two USB micro-B connectors.

2.1.1 Hardware configuration to run the demonstration firmware

Table 1 hereafter lists the proper jumper configuration to run the demonstration firmware on the 32L496GDISCOVERY board.

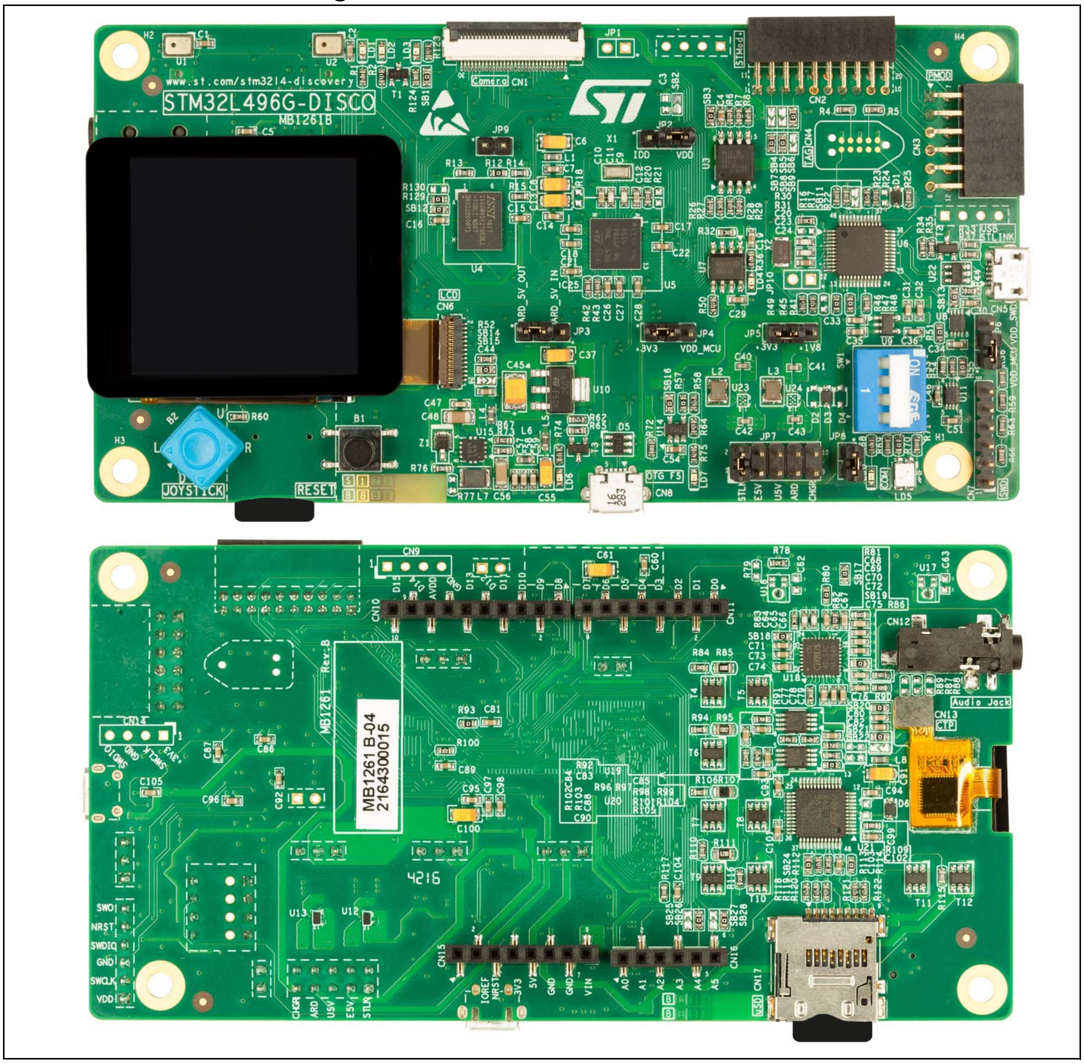
Table 1. Jumpers configuration

Jumper/connector number	Position (note)
JP2	1-2 (IDD)
JP3	2-3 (ARD_V5_IN)
JP4	1-2(+3V3)
JP5	1-2(+3V3)
JP6	1-2
JP7	STLK
JP8	CLOSED
JP9	OPENED

Note: Position 1 corresponds to jumper side with a dot marking.

Refer to UM2160 Discovery board with STM32L496AGT6 MCU for complete description of jumper settings.

Figure 2. 32L496GDISCOVERY board

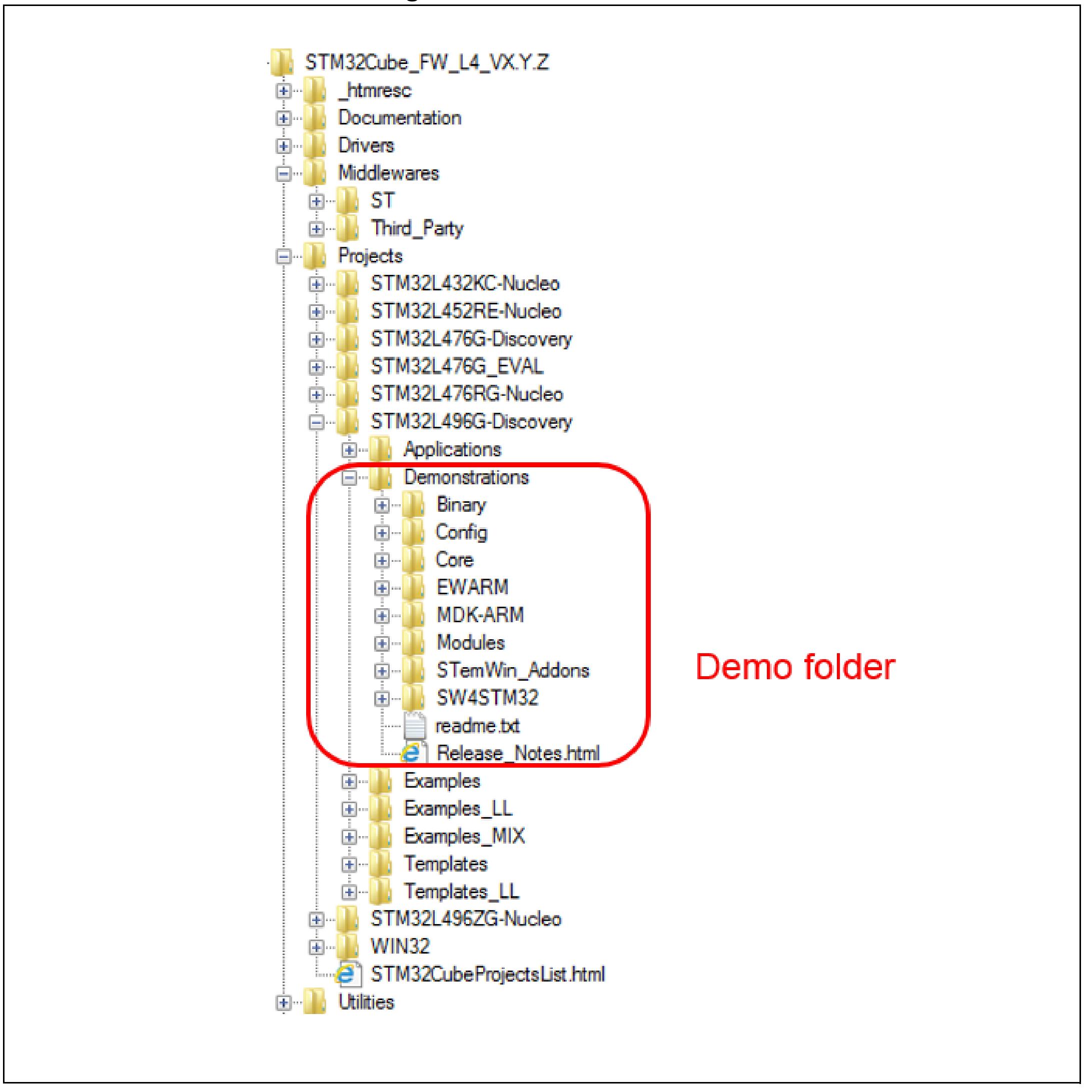


3 Demonstration firmware package

3.1 Demonstration repository

The STM32CubeL4 demonstration firmware for 32L496GDISCOVERY board is provided within the STM32CubeL4 firmware package as shown in *Figure 3*.

Figure 3. Folder structure



The demonstration sources are located in the projects folder of the STM32Cube package for each supported board. The sources are divided into five groups described as follows:

- **Binary**: demonstration binary file in Hex format
- **Config**: all middleware components and HAL configuration files
- **Core**: contains the kernel files
- **Modules**: contains the sources files for main application top level and the application modules.
- **Project settings**: a folder per tool chain containing the project settings and the linker files.

3.2 Demonstration architecture overview

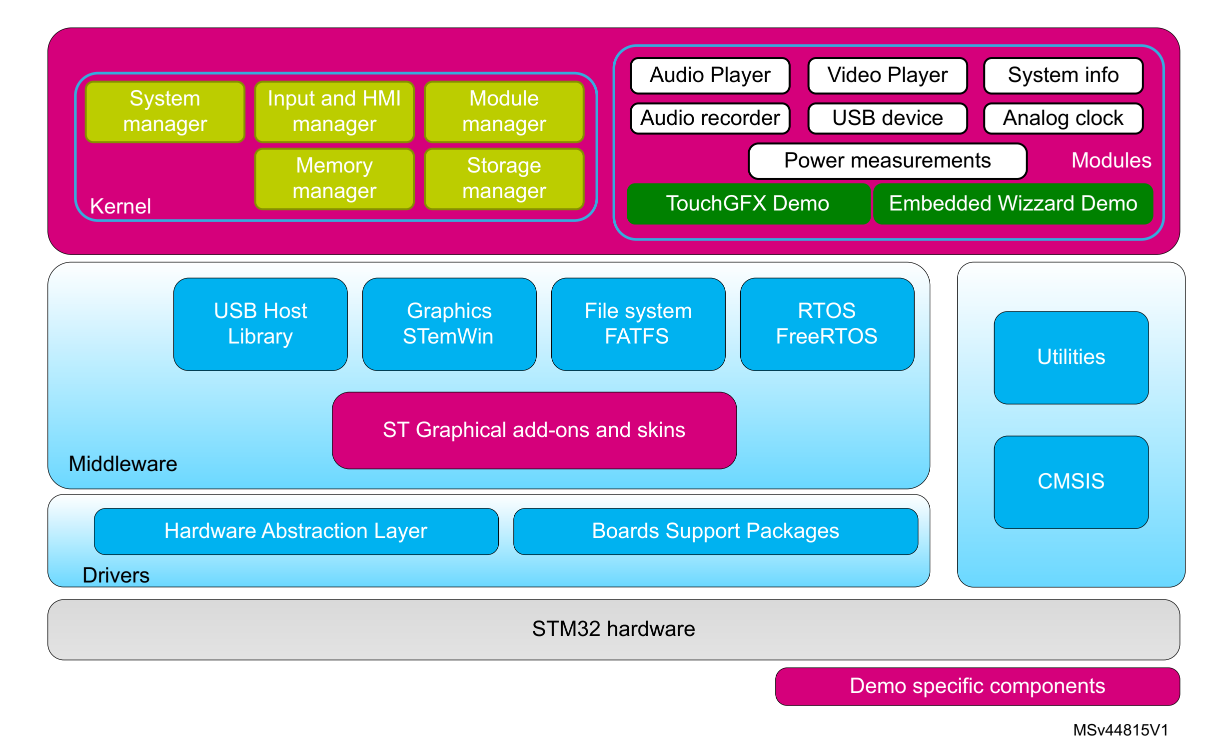
The STM32CubeL4 demonstration firmware for 32L496GDISCOVERY board is composed of a central kernel based on a set of firmware and hardware services offered by the STM32Cube middleware, evaluation board drivers and a set of modules mounted on the kernel and built in a modular architecture.

Each module can be reused separately in a standalone application. The full set of modules is managed by the Kernel which provides access to all common resources and facilitates the addition of new modules as shown in *Figure 4*.

Each module provide the following functionalities and proprieties:

1. Display characteristics
2. Method to startup the module
3. Method to close down the module for low power mode
4. The module application core (main module process)
5. Specific configuration
6. Error management

Figure 4. Demonstration architecture overview

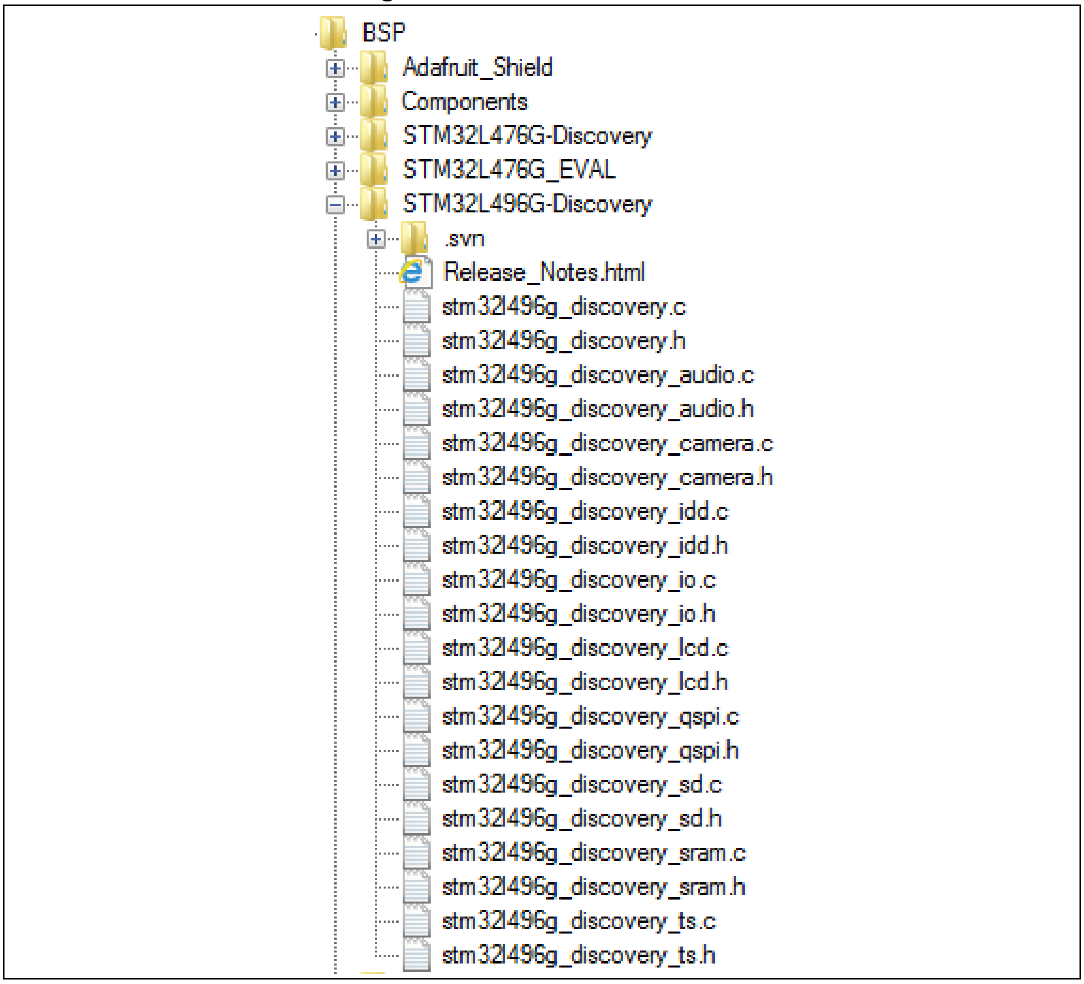


Kernel services are described in *Section 4.1: Kernel*.

3.3 32L496GDISCOVERY board BSP

Board drivers are available within the stm32l496g_discovery_XXX.c/.h files (see *Figure 5*), implementing the board capabilities and the bus link mechanism for the board components (LEDs, Buttons, audio, LCD, external PSRAM and QuadSPI flash memories, Touch Screen, microSD card, digital camera interface)

Figure 5. EVAL BSP structure



Components present on the 32L496GDISCOVERY board are controlled by dedicated BSP drivers. These are:

- The IO expanders in *stm32l496g_discovery_io.c/.h*
- The CS42L51 audio codec with independent audio content in *stm32l496g_discovery_audio.c/.h*
- The 8-Mbit PSRAM memory in *stm32l496g_discovery_sram.c/.h*
- The 64-Mbit Macronix MX25R6435F Quad-SPI flash memory in *stm32l496g_discovery_qspi.c/.h*
- The microSD card in *stm32l496g_discovery_sd.c/.h*
- The built-in Idd circuitry for MCU current consumption measurement in *stm32l496g_discovery_idd.c/.h*
- The 1.54-inch 240x240 dot-matrix color LCD panel with resistive touchscreen in *stm32l496g_discovery_lcd.c/.h* and *stm32l496g_discovery_ts.c/.h*
- -The digital camera interface *stm32l496g_discovery_camera.c/.h*

4 Demonstration functional description

4.1 Kernel

The demonstration is built around the STemWin Graphical Library, based on SEGGER emWin. STemWin is a professional graphical stack library, enabling Graphical User Interfaces (GUI) building up with any STM32, any LCD and any LCD controller, taking benefit from STM32 hardware accelerations, whenever possible.

Two other graphic demonstrations are provided and delivered in binary format

- Embedded Wizard by Tara Systems
- Touch GFX by Draupner Graphics (two versions provided: a complete and a light version)

The graphical aspect of the STM32CubeL4 demonstration is divided into several graphical components:

- the startup window (*Figure 6*) showing the progress of the hardware and software initialization;
- the main desktop (*Figure 7*) that yields access to four graphic demos
  - STemWin graphic demo
  - Embedded Wizard graphic demo
  - TouchGFX graphic demo (light)
  - TouchGFX Full graphic demo

Figure 6. Start-up window

Figure 7. Main desktop screen

By pressing on the proper icon, the user starts the associated graphic demo.

STemWin, Embedded Wizard and TouchGFX Lite are readily available to the user: when the corresponding icon is pressed, the demonstration immediately starts.

For QuadSPI size limitation reasons, the TouchGFX full demo first automatically downloads code and graphic resources from the SD card to the embedded Flash and to the QuadSPI. This operation overwrites the Emebdded Wizard and TouchGFX light demonstration resources present by default.

The download is indicated by a warning message and a progress bar. When it is over, TouchGFX Full demonstration starts.

User can come back to Embedded Wizard and TouchGFX light demo. Launching either one requires another download operation to restore the graphic resources, either that of Embedded Wizard or TouchGFX light.

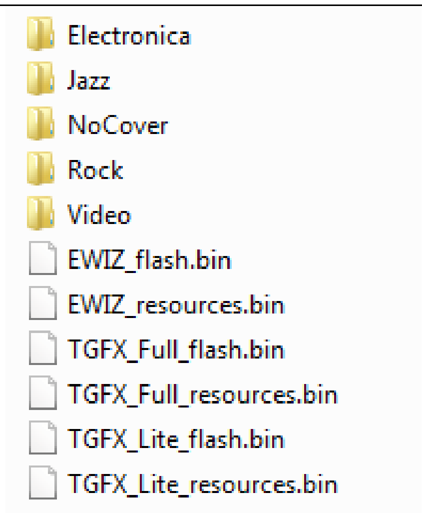
4.1.1 SD card compulsory usage and content

The download process mentioned here above makes the use of the SD card while running the demonstration compulsory.

The SD card (that comes with the Discovery board) contains the binaries and the graphic resources that are downloaded upon request from the user. Additionally, the SD card contains the video and audio files played by the demonstrations.

The folder hierarchy to be used is fixed and shown in *Figure 8*.

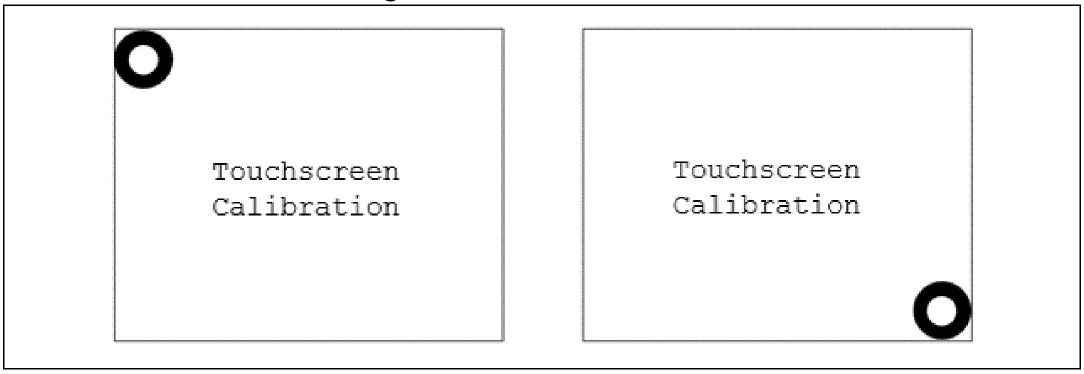
Figure 8. SD card contents and folders hierarchy



4.1.2 Touchscreen calibration

When the demonstration is launched for the very first time, the touchscreen needs to be calibrated. To do this, before the startup screen is displayed, the user has to follow the displayed calibration instructions by touching the screen at the indicated positions (*Figure 9*). This allows to get the physical Touch screen values that will be used to calibrate the screen.

Figure 9. Calibration screens



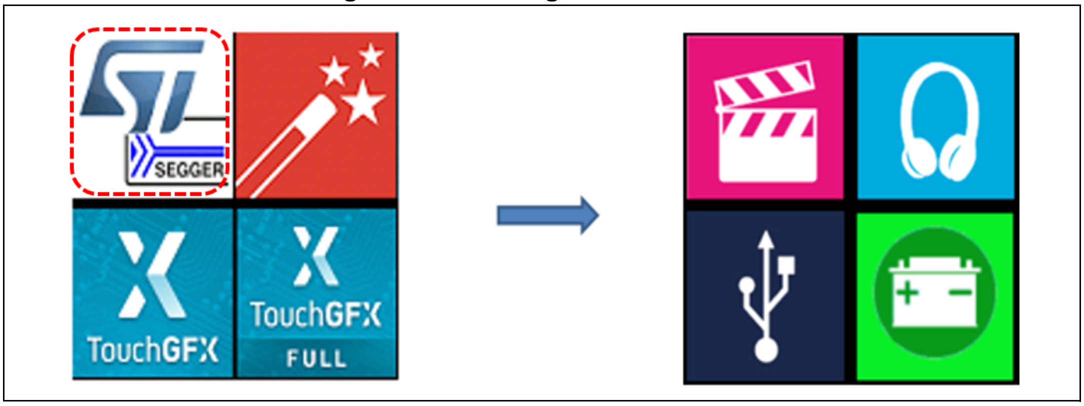
To calibrate again the touchscreen, user needs to press the joystick SEL button while powering on the board or during a software reset (e.g., during a RESET button press).

More information on the calibration process is provided *Section 8.2*.

4.2 STemWin graphic demonstration modules

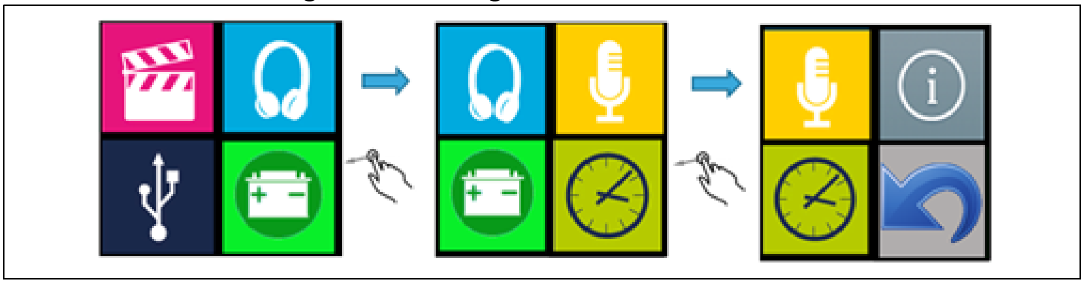
Pressing STemWin icon allows to enter STemWin demo as indicated in *Figure 10*.

Figure 10. Entering STemWin demo



STemWin offers several modules, four of them being displayed per screen. User can easily move from one to another in sliding left or right. Display moves column-wise (left or right) as shown in *Figure 11* where two consecutive screen left moves are simulated

Figure 11. Moving thru STemWin modules



4.2.1 Audio Player

Overview

The audio player module provides a complete audio solution based on the STM32L496AGT6 MCU and delivers a high-quality music experience. It supports playing music in WAV format but may be extended to support other compressed formats such as MP3 and WMA audio formats.

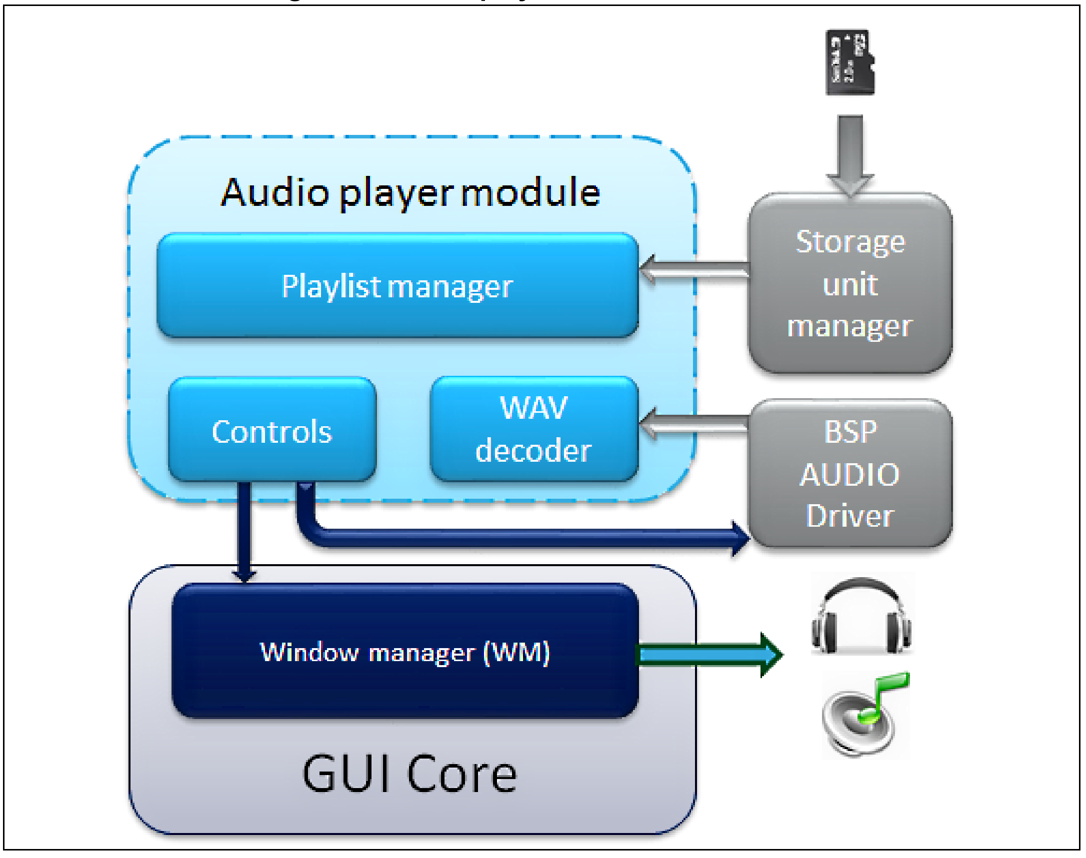
Features

- Audio format: WAV format without compression, with 8 k to 96 k sampling
- Audio files stored in SD Card
- Only 8 Kbytes of RAM required for audio processing

Architecture

Figure 12 shows the different audio player parts and their connections and interactions with the external components.

Figure 12. Audio player module architecture



Process description

The audio player initialization is done in the startup step. In this step, all the audio player states, the speaker and the volume value are initialized. When the play button in the audio player interface is pressed, the audio process is started. Start the audio player module from the main desktop menu as shown in *Figure 13*.

Figure 13. Audio player module architecture

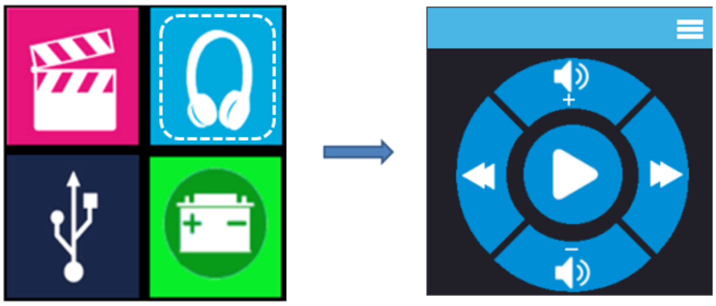
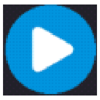
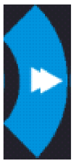
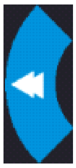
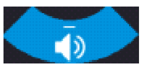


Table 2. Audio player module controls

Button	Preview	Description
Play		Reads the wave file from storage unit Starts or resumes the audio task Starts playing audio stream Replaces Play button with Pause button
Next		Points to the next audio file Stops audio playing Starts playing the next audio file if Play button is pressed
Previous		Points to the previous audio file Stops audio playing Starts playing the previous audio file if Play button is pressed
Volume up		Increases the volume
Volume down		Decreases the volume
Exit		Closes the module

4.2.2 Audio Recorder

Overview

The audio recorder module can be used to record audio frames in WAV format, save them in the storage unit, and play them afterwards. Audio input can either be the headset microphone or the Discovery board microphones.

Features

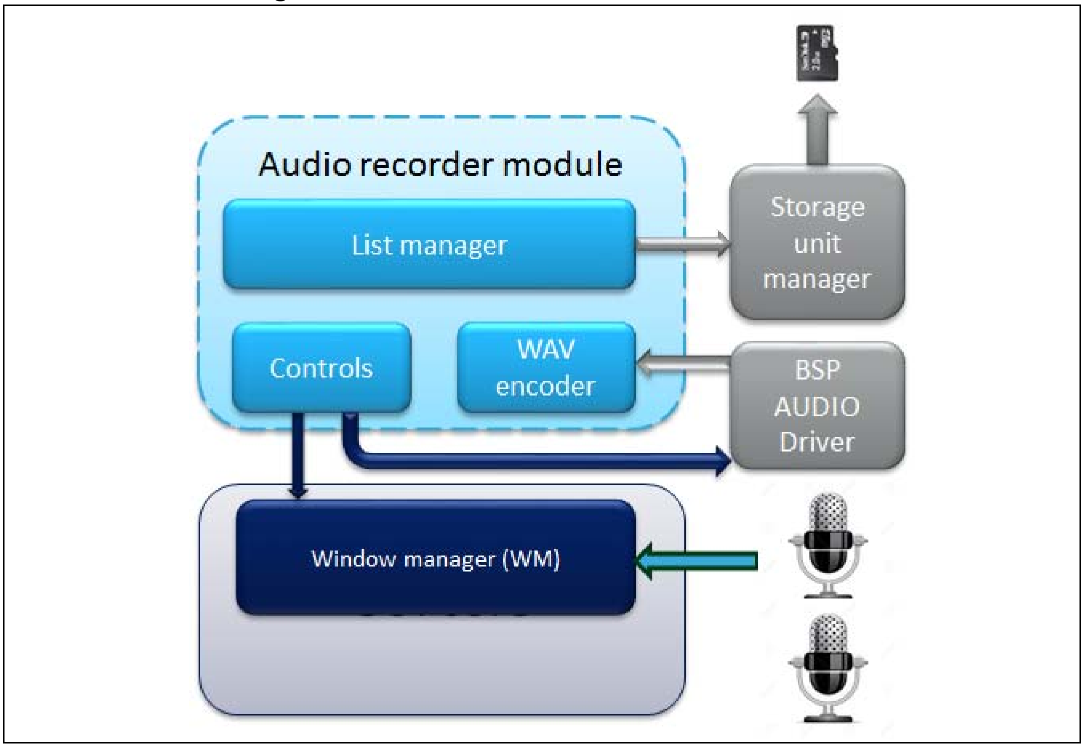
- Audio format: WAV format without compression, with 16 k sampling stereo
- Recorded files stored in SD Card
- Embeds quick audio player
- Only 8 Kbytes of RAM required for audio processing

The MP3 format is not supported, but can be easily added (separate demonstration).

Architecture

Figure 14 shows the different audio recorder parts and their connections and interactions with the external components

Figure 14. Audio recorder module architecture



Functional description

Start audio recorder module by touching the audio recorder icon, as indicated in *Figure 15*.

Figure 15. Audio recorder module start-up

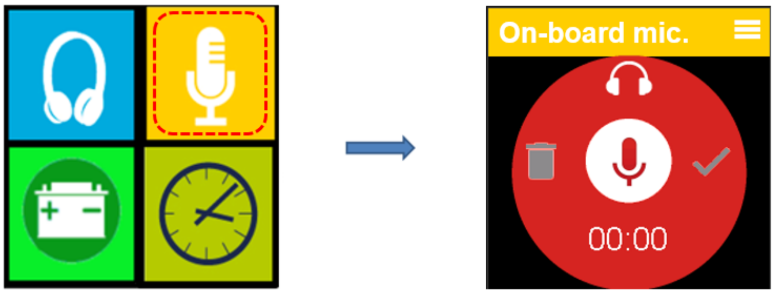


Table 3. Audio recorder module controls

Button	Preview	Description
Record		Starts recording audio Replaces record button by pause button
Play		Reads the recorded wave file from the storage unit
Pause		Suspends the audio task Pauses the audio file record
Save		Saves the recorded file in the storage unit Suspends the audio task Stops audio recording
Remove		Stops audio recoding Discards the recorded wave
Change audio input	 	Allows to either use board microphones (left hand side icon) or Headset microphone (right hand side icon)
Exit		Closes the module

4.2.3 Video module

Overview

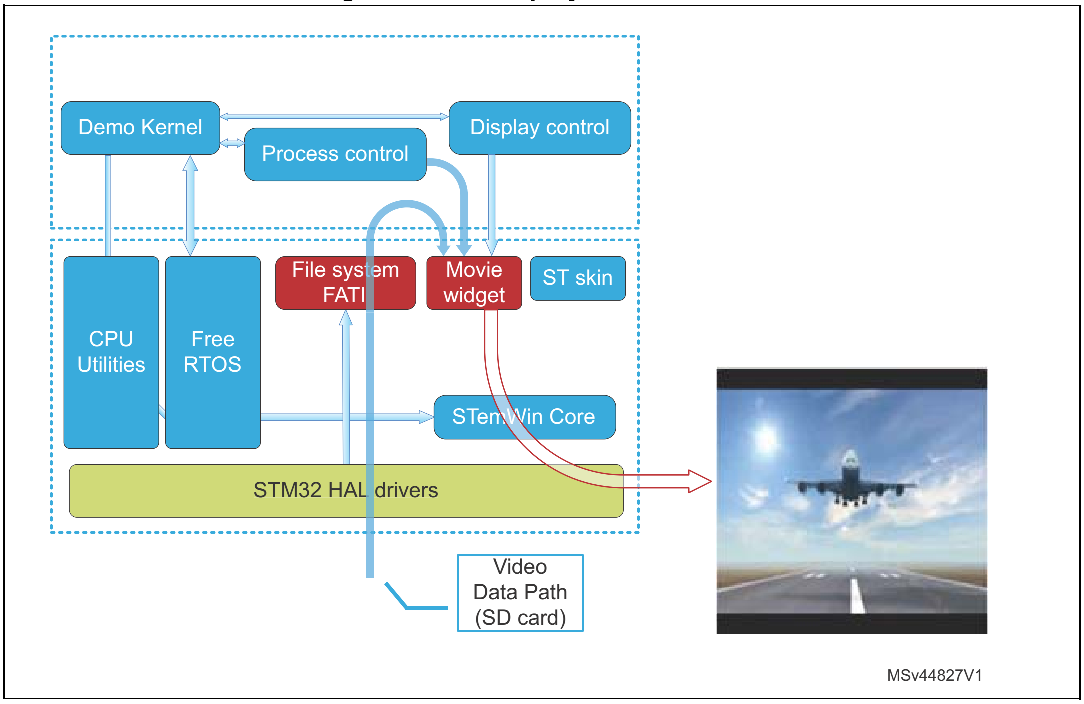
The video player module provides a video solution based on the STM32L4xxx and the STemWin movie APIs. It supports the AVI format.

Features

- Video Format: AVI
- Performance: frame rate up to 13 fps
- Video files stored in SD Card

Architecture

Figure 16. Video player architecture

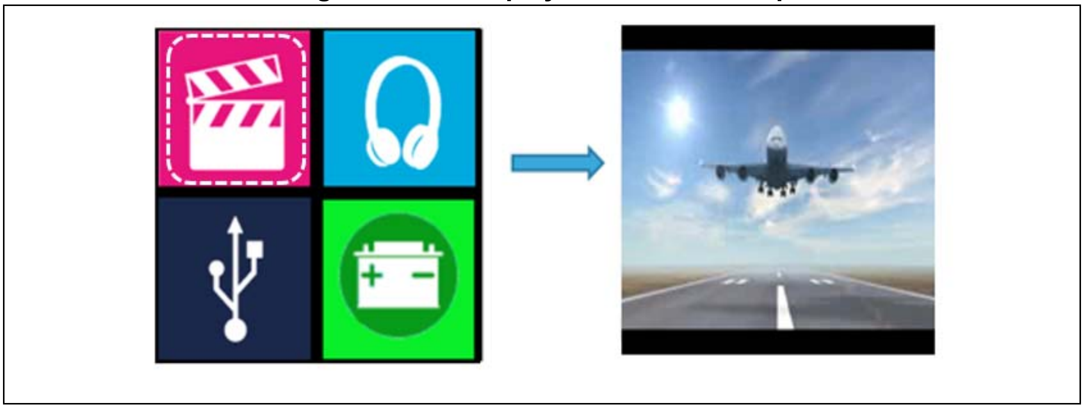


Functional description

Start video player module by touching the video player icon, as indicated in *Figure 17*.

When the video player is started, the first AVI file stored in the storage unit starts playing.

Figure 17. Video player module start-up



4.2.4 Analog clock module

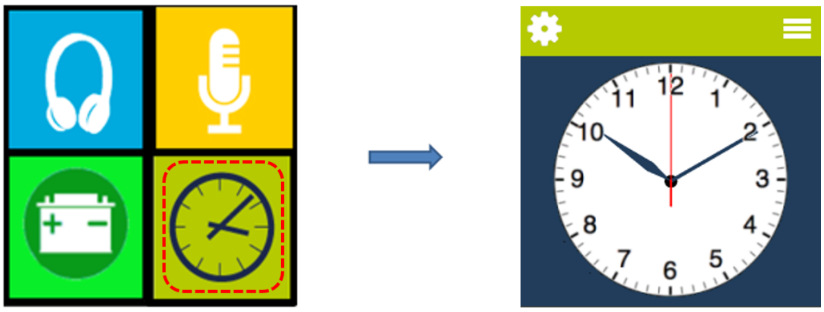
Overview

The analog clock module enables to show and adjust the analog time by changing the RTC configuration.

Functional description

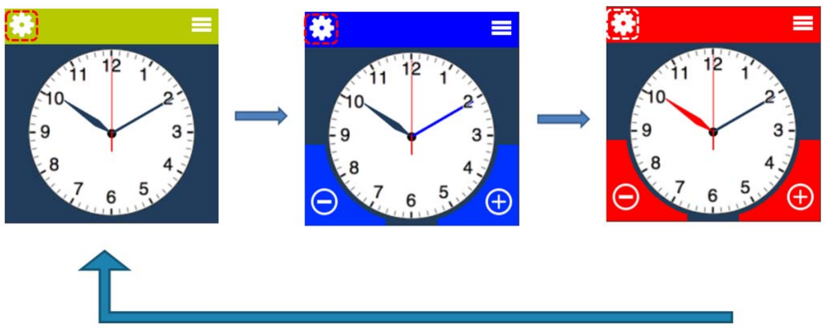
1. Start analog clock module by touching the analog clock icon (see *Figure 18*).

Figure 18. Analog clock module start-up



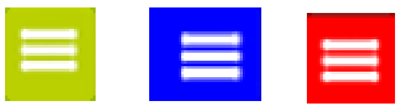
2. Press on settings button; first to set minutes, then to set hours (*Figure 19*). “+” and “-“ buttons respectively allow to move clockwise and anticlockwise.

Figure 19. Analog clock setting



3. Pressing the exit button in any sub mode allows to exit the module

Figure 20. Analog clock menu exit buttons



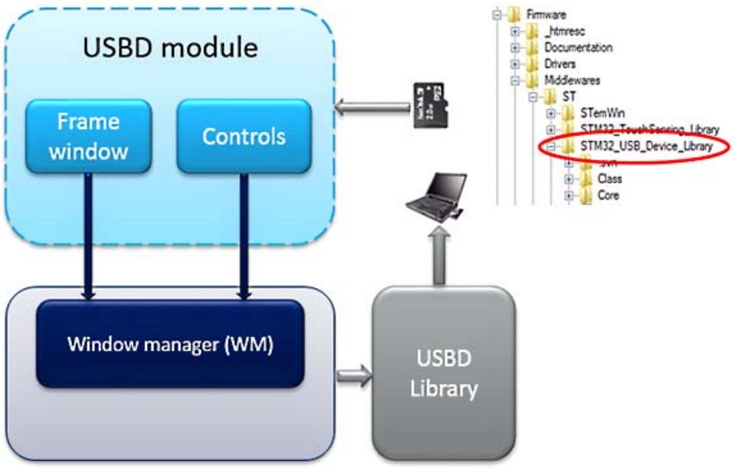
4.2.5 USB devices module

Overview

The USB device (USBD) module includes mass storage device application using the Micro SD memory.

Figure 21 shows the different USB modules, and their connections and interactions with the external components.

Figure 21. USBD module architecture



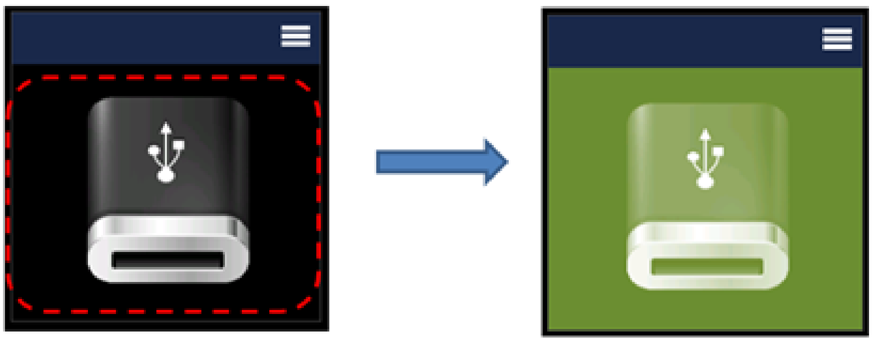
Functional description

1. Start USBD module by touching the USB device icon (see *Figure 22*). A USB type A to micro-B cable must be connected from a PC to CN8 connector.

Figure 22. USBD module startup

2. Connect the USB device by touching the screen (except the header zone) (*Figure 23*).

Figure 23. Connection of an USB device



The board now behaves as a USB device.

Pressing the exit button allows to exit the module.

Figure 24. USB menu exit button

4.2.6 IDD

Overview

The IDD module application measures and displays in real time the MCU current consumption depending on the selected power mode. The current is measured and calculated thanks to a second microcontroller on the board which is a STM32 L1 MCU.

To access the IDD module, the user must touch the power measurements icon as shown in *Figure 25*.

Figure 25. Entering power consumption measurements menu

From that point, the user can slide right (or left to move back) to go over all the possible low power modes that are proposed:

- Run mode at 24Mhz (voltage range 2), PLL off, RTC/LSE off, Flash ART on
- Sleep mode at 24Mhz (voltage range 2), PLL off, RTC/LSE off, Flash ART on
- Low Power Run mode at 2Mhz, PLL off, RTC/LSE off, Flash ART on
- Low Power Sleep mode at 2Mhz, PLL off, RTC/LSE off, Flash ART on
- Stop 2 mode, RTC/LSE off, Flash ART off
- Standby mode, RTC/LSE off, Flash ART off, RAM retention off
- Shutdown mode, RTC/LSE off, Flash ART off

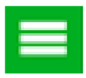
Figure 26 shows the succession of the low power modes.

Figure 26. Moving thru low power modes sub-menus

User initiates the power consumption measurement in pressing the “Activate” button. LCD is turned off in all cases except in Run mode then result is displayed after a couple of seconds.

Pressing the exit button allows to exit the module.

Figure 27. Power consumption menu exit button



4.2.7 System Information

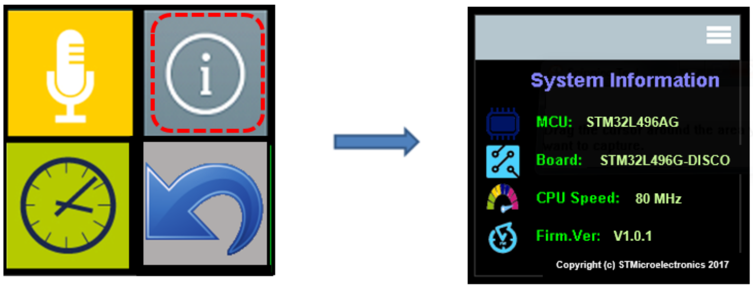
Overview

The system information shows the demonstration information such as:

- The MCU,
- The used board,
- The CPU speed,
- The FW version

To access the Information module, the user must touch the information icon as shown in *Figure 28*.

Figure 28. Entering information menu



Pressing the exit button allows to exit the module.

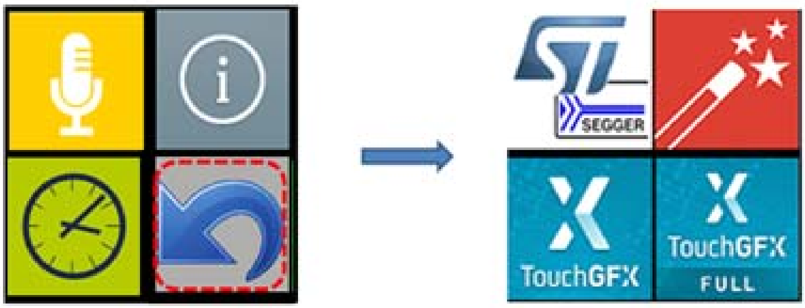
Figure 29. Information menu exit button

4.2.8 STemWin demo exit

Overview

The remaining icon in the STemWin demo allows to come back to the main desktop as indicated in *Figure 30*. Pressing this icon triggers a software reset. It allows to quit the STemWin demo to choose to run the same or another one.

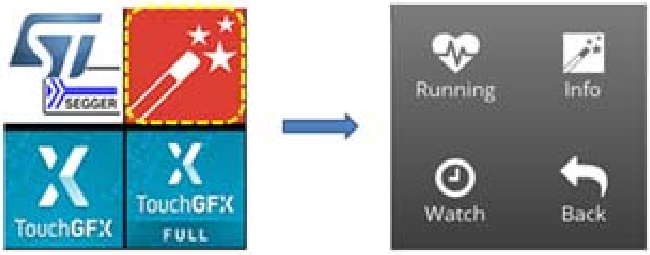
Figure 30. Exiting STemWin demo



4.3 Embedded Wizard graphic demonstration modules

Pressing Embedded Wizard icon allows to enter Tara Systems demo (see *Figure 31*).

Figure 31. Entering Embedded Wizard demo



Four menus are proposed, that user can access by simple icon press.

4.3.1 Watch

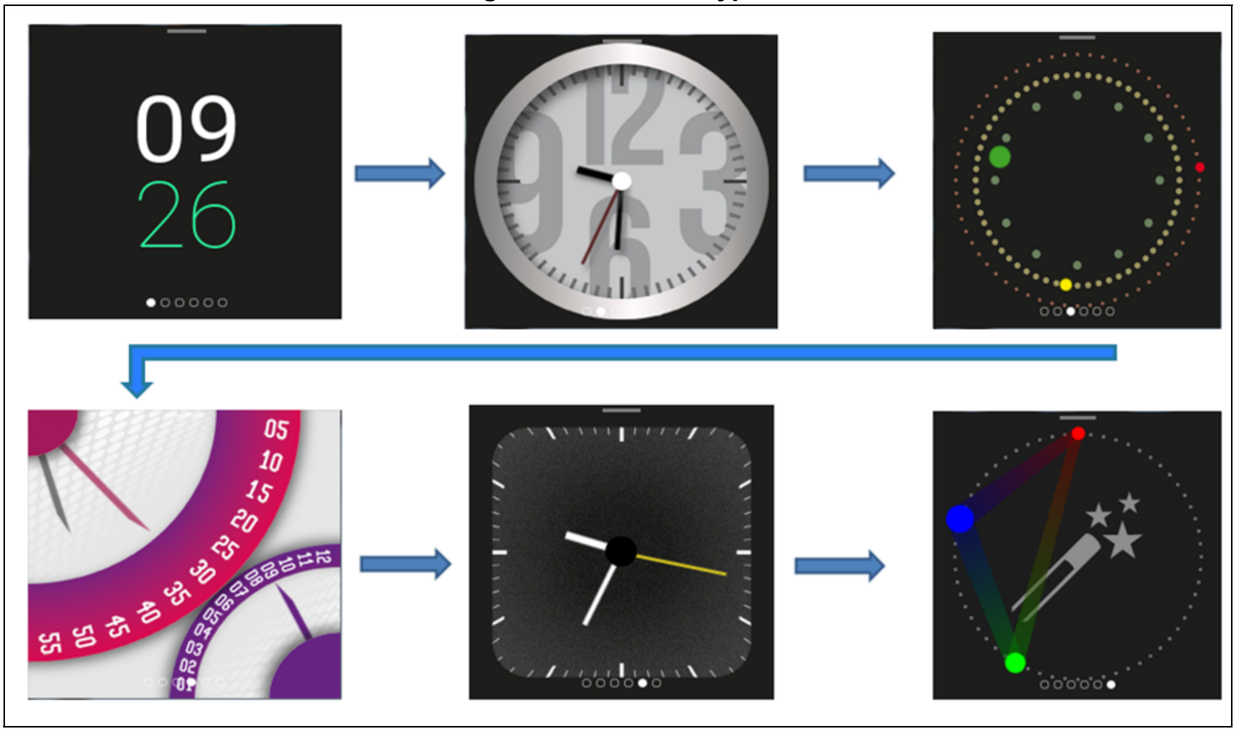
Overview

The watch menu yields several types of watches that can be designed thru Embedded Wizard. Watch menu is entered in pressing the Watch icon (see *Figure 32*). User can browse thru the different watches in sliding the screen left or right.

Figure 32. Entering Watch menu

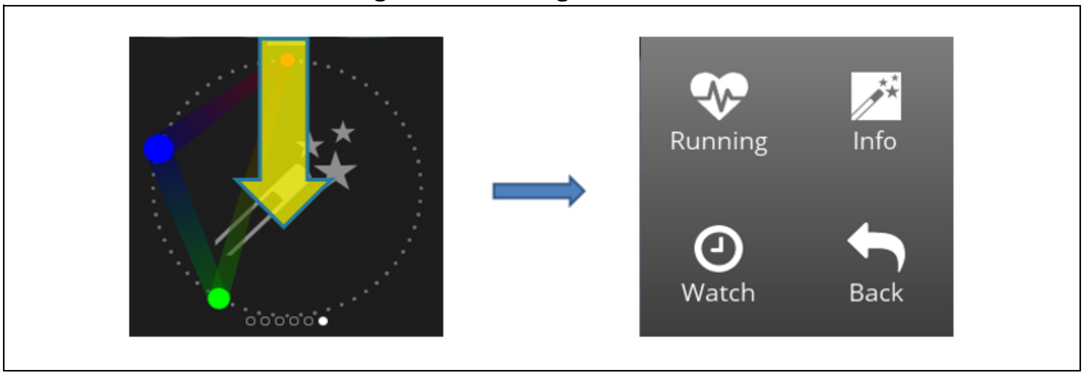
Figure 33 shows the different watches types when sliding from right to left. Watches time is initialized to a default value and the user can see the time ticking based on the MCU clock.

Figure 33. Watches types



Exit watches menu in sliding down the screen from up to bottom, similarly to pulling down a curtain as shown in *Figure 34*. Exiting is possible from any window.

Figure 34. Exiting Watch menu



4.3.2 Running

Overview

Running menu provides different displays that can be used by an activity monitoring feature. This feature is simulated (no sensors are connected to the Discovery Board) and the different sub-menus are:

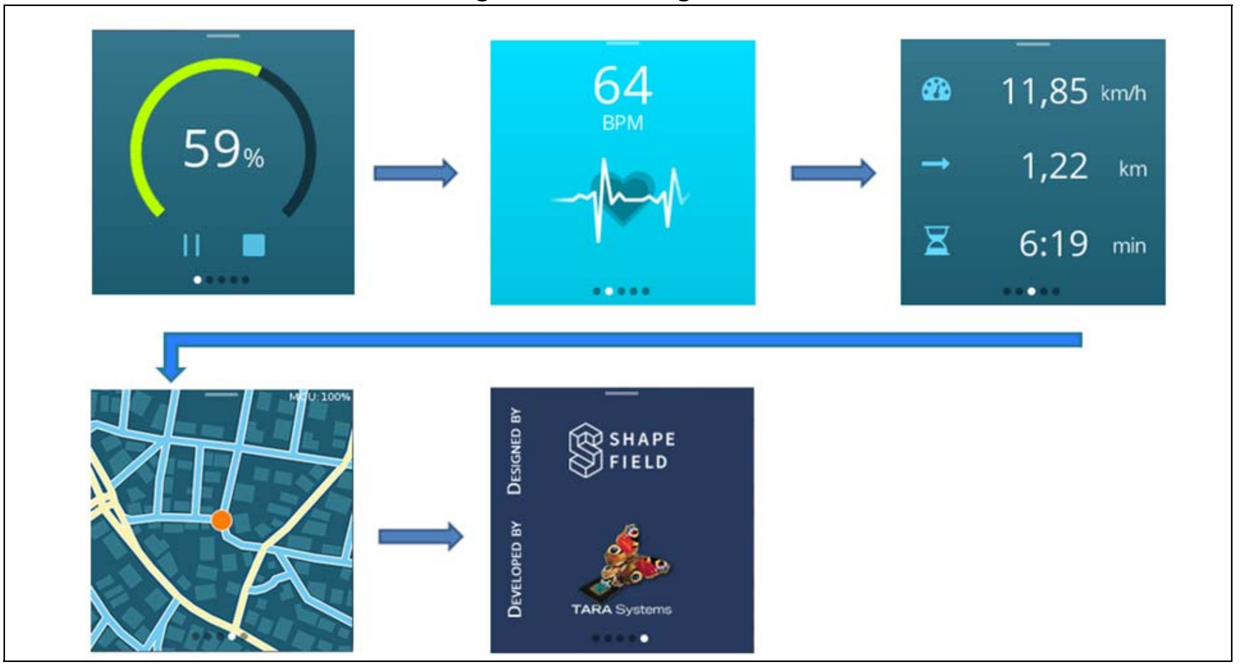
- A counter
- A heart-rate monitor
- A “dashboard” yielding the speed, the distance and the elapsed time
- A map where a red dot is moving on the fly
- ShapeField signature

Running menu is entered in pressing the Running icon as shown in *Figure 35*.

Figure 35. Entering Running menu

Figure 36 shows the different sub-menus listed here-above when sliding from right to left. Similarly to the Watch menu, the figure in the upper right corner provides the CPU load. The user can navigate from one menu to another in sliding left or right.

Figure 36. Running menus



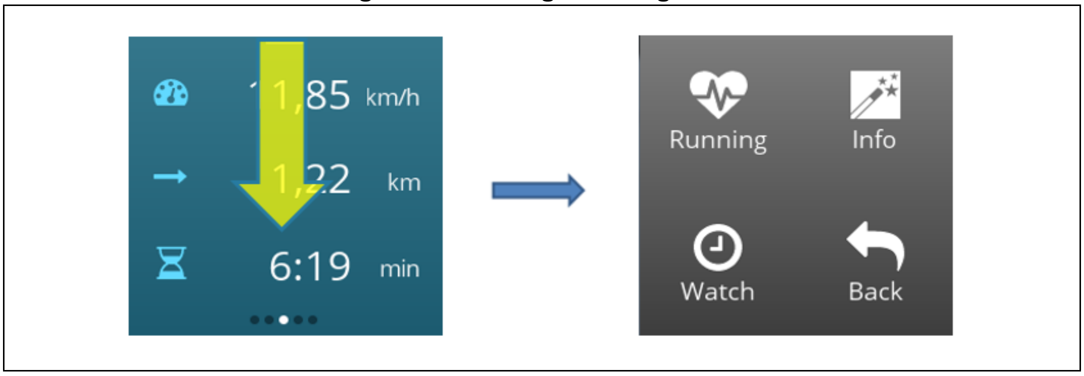
The map sub-menu shows at the top right corner a figure providing the CPU load in real time. This figure allows to underline the gain provided by the Chrom-ART (DMA2D) hardware IP that can be enabled or disabled by the user as described hereafter.

The first display allows to start, stop, resume or reset the animation as illustrated by *Figure 37*.

Figure 37. Managing Running animation

Exiting the running menu can be done from any sub-menu display, the same way as for the Watch menu: the user just needs to slide down the screen from up to bottom, as shown in *Figure 38*.

Figure 38. Exiting Running menu



4.3.3 Info

Overview

Info menu yields several displays each providing some short information on “Embedded Wizard”. Info menu is entered in pressing the Info icon as shown in *Figure 39*.

Figure 39. Entering Info menu

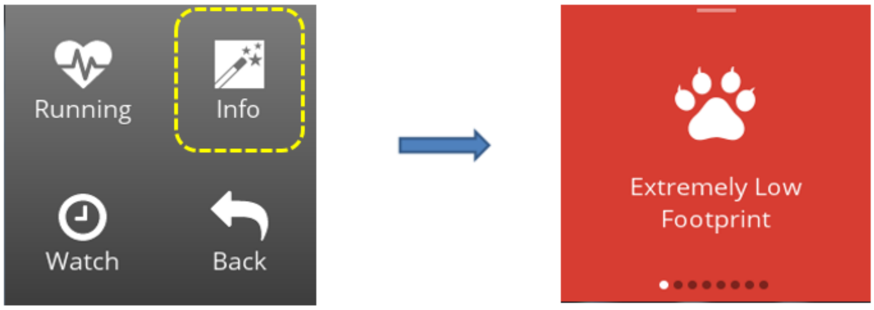
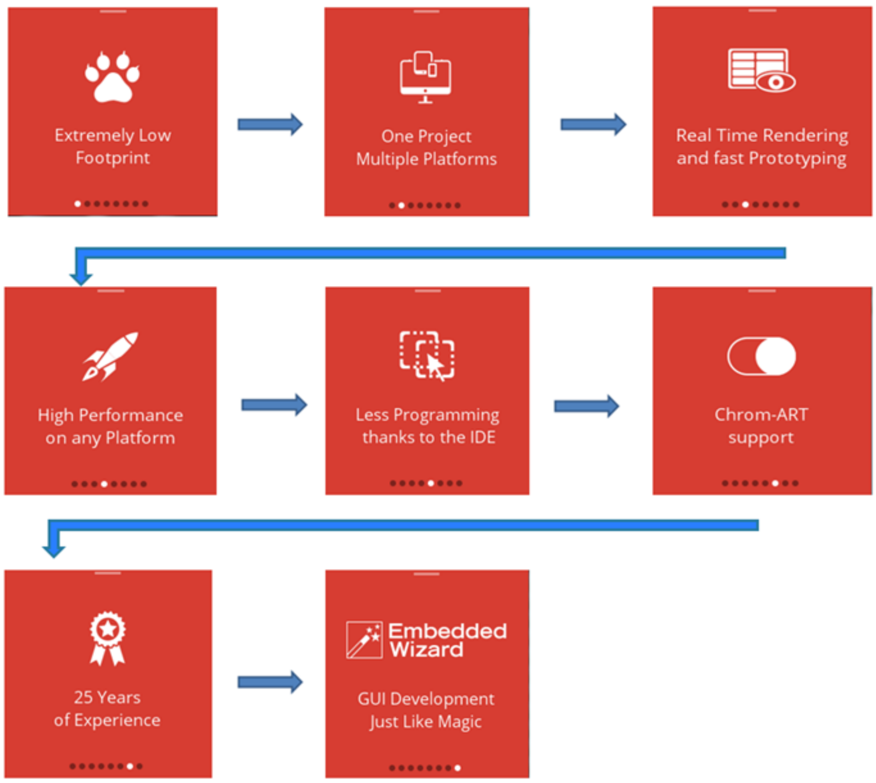


Figure 40 shows the different Info sub-menus when sliding from right to left. The user can navigate from one menu to another in sliding left or right.

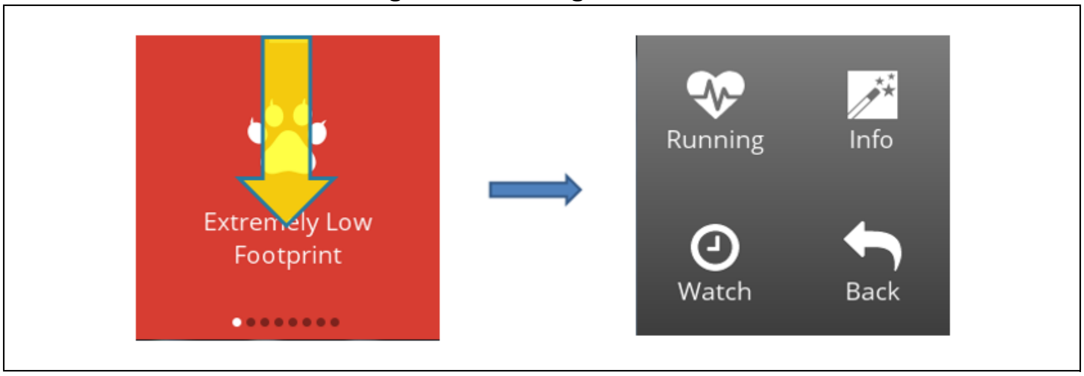
One of those sub-menus is that of the DMA2D / Chrom-ART IP. Further details are provided in the *Section 4.3.4: Enabling/Disabling Chrom-ART (DMA2D)*.

Figure 40. Info menus



Exiting the Info menu can be done from any sub-menu display, the same way as for the other menus: the user has to slide down the screen from up to bottom, as shown in *Figure 41*.

Figure 41. Exiting Info menu



4.3.4 Enabling/Disabling Chrom-ART (DMA2D)

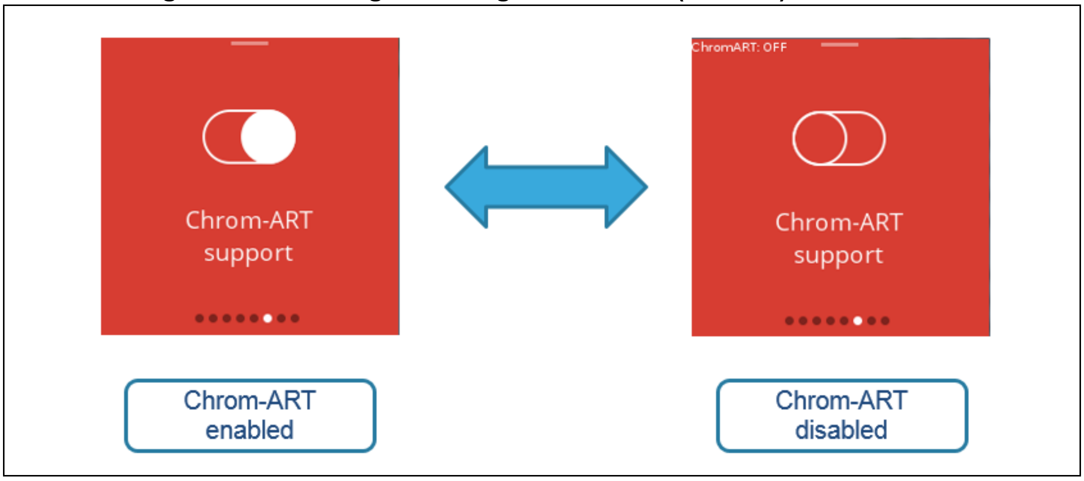
Overview

Embedded Wizard demo allows to enable or disable the Chrom-ART (DMA2D) hardware IP for all the menus. This option is accessible thru the Info menu Chrom-ART support display.

The hardware IP can be disabled by the user by a mere left slide of the button on the screen. The IP can be re-enabled by a right slide of the same button. This feature is illustrated in *Figure 42*.

The user can check on the map sub-menu of the Running demo (*Section 4.3.2*) the CPU load large increase when the Chrom-ART hardware IP is disabled. This underlines the gain the hardware IP is providing for all the actions related to image display

Figure 42. Enabling/Disabling Chrom-ART (DMA2D) hardware IP

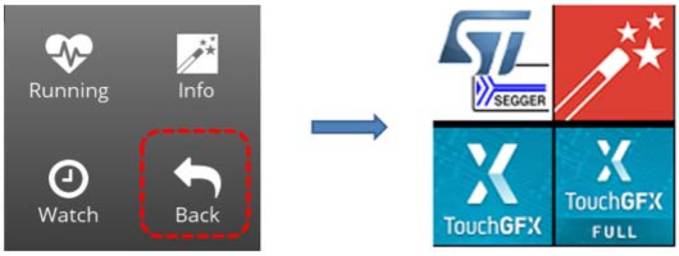


4.3.5 Embedded Wizard demo exit

Overview

The last icon in the Embedded Wizard demo allows to come back to the main desktop as indicated in *Figure 43*. Pressing this icon triggers a software reset. It allows to quit the Embedded Wizard demo to choose to run the same or another one.

Figure 43. Exiting Embedded Wizard demo



4.4 TouchGFX graphic demonstration modules

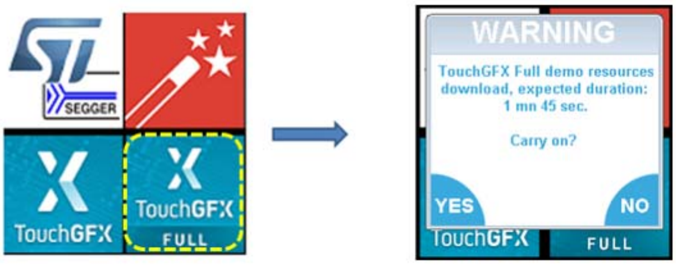
4.4.1 Lite versus Full TouchGFX demos

Overview

As stated in *Section 4.1*, STemWin, Embedded Wizard and TouchGFX Lite are readily available to the user: when the corresponding icon is pressed, the demonstration immediately starts.

For Quad-SPI size limitation reasons, the TouchGFX full demo first automatically downloads code and graphic resources from the SD card to the embedded Flash and to the Quad-SPI. As shown in *Figure 44*, a warning message pops up before the actual download in case the user prefers to cancel the request.

Figure 44. Starting TouchGFX Full demo



Once ToughGFX Full demo is loaded, there is no more any download operation to start it again. However, to start Embedded Wizard or TouchGFX Lite, a new download operation is requested and the same warning message pops up at request time (with the appropriate download time).

Lite and Full TouchGFX demos yield the same menus: audio player, game, watches displays, activity monitoring simulation. The difference merely lies in the number of graphic items (number of watches displays, graphic effects when the audio player is running,…) that makes the Full TouchGFX demo more demanding in resources.

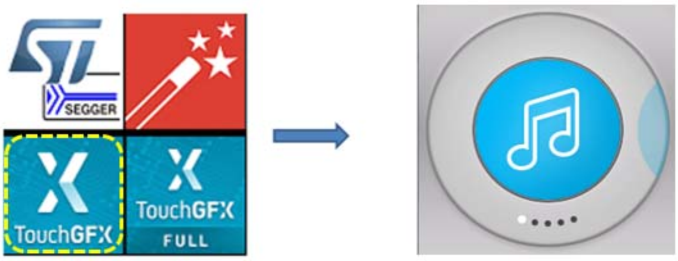
In *Section 4.4.2*, the descriptions will be applicable to both the Lite and the Full demos since the menus are the same. When needed, the difference between the Lite and the Full demos will be highlighted.

4.4.2 Entering TouchGFX demo

Overview

Entering Lite or Full TouchGFX demo is achieved in pressing the correspond icon. *Figure 45* shows the example of the Lite demo, assuming no download operation is required. The upcoming icon is that of the first sub-demo (the audio player).

Figure 45. Entering TouchGFX menu



4.4.3 TouchGFX menus

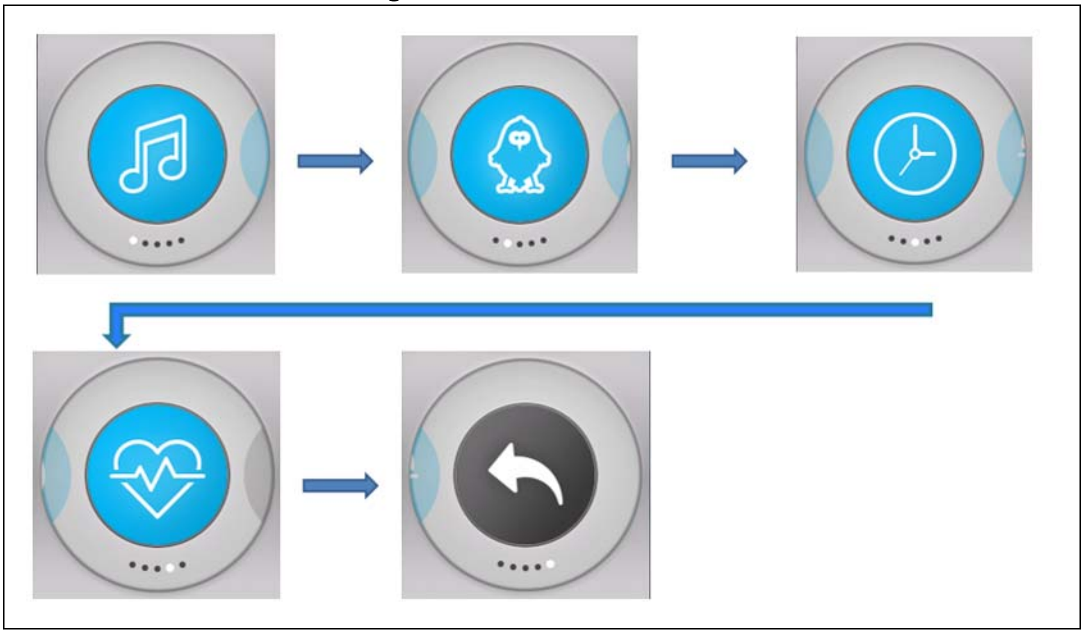
Overview

TouchGFX different menus are only accessible one at a time. The user must slide the screen left or right to access to the desired one. *Figure 46* displays the menus icons as they show up when sliding the screen from right to left.

The menus are, from left to right and from top to bottom,

- The audio player,
- The bird game,
- The watches displays,
- The activity monitoring simulation,
- The exit button

Figure 46. TouchGFX menus



Entering a sub-menu, whatever it is, is simply achieved in touching the icon. This action will not be detailed in *Section 4.4.4*.

4.4.4 Audio player

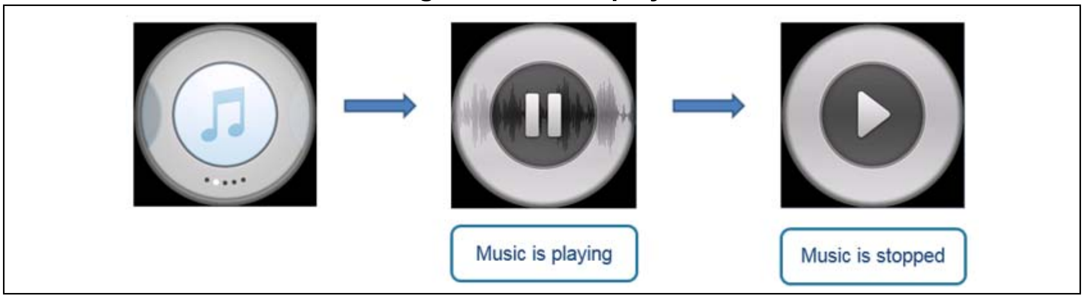
Overview

The first display that shows up is that of a musical note; dots below the note yield the number of musical albums present on the SD card. Navigating from one album to another is achieved in sliding the screen left or right.

1. **Audio playing**

To start playing music, the user needs to press the musical note or to slide the screen from top to bottom. Playing starts, the user can stop the music at any time in pressing the stop button. Resuming the music is done in pressing the start button. *Figure 47* illustrates the different actions.

Figure 47. Audio player



Note: The acoustic waveform showing up when music is playing is available only in the Full demo.

2. **Volume setting**

Volume is adjusted by moving the finger circularly on the right hand side of the icon.

– Volume up: clockwise direction
– Volume down: clockwise direction

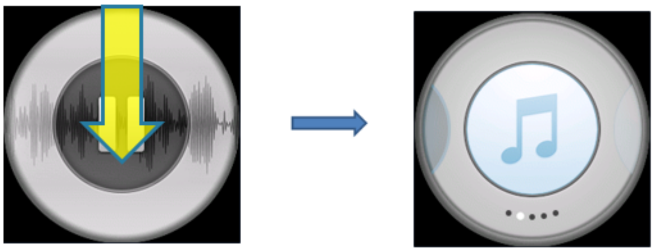
Figure 48 illustrates this setting.

Figure 48. Audio player volume setting

3. **Playing mode exit**

To exit the playing stage, the user must slide the screen from up to bottom as shown in *Figure 49*.

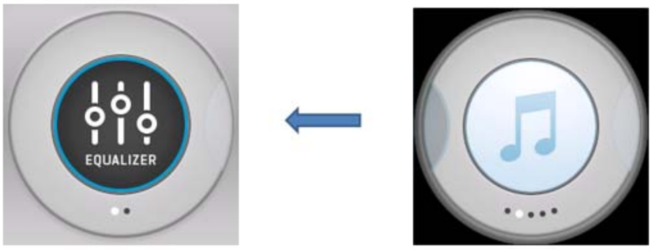
Figure 49. Exiting playing mode



4. **Equalizer setting**

The equalizer menu is accessed from the audio player initial display by a left to right screen slide (see *Figure 50*).

Figure 50. Accessing the equalizer menu



The equalizer divides the audio bandwidth in 5 bands: bass, low, mid, upper and high. To select the band on which to apply the desired amplification, the user has to slide horizontally the line where the bands are listed and to stop on the correct one as described by *Figure 51*.

Figure 51. Equalizer audio band selection

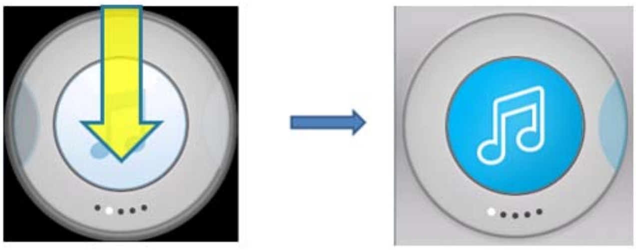
Once the band is selected, the blue button can be moved thru the touch screen to increase or decrease the amplification as shown in *Figure 52*.

Figure 52. Equalizer audio band amplification setting

5. Exiting audio player menu

To exit the audio player menu, the user must slide the screen from up to bottom as shown in *Figure 53* from the equalizer sub-menu or any audio volume selection menu.

Figure 53. Exiting the audio player

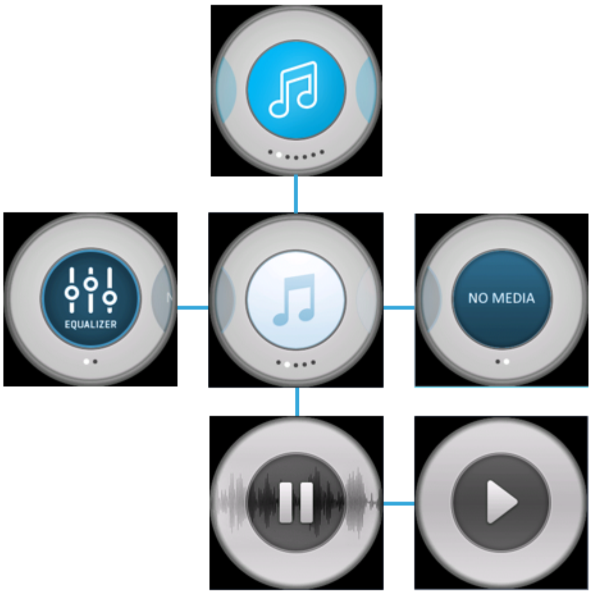


6. Exiting audio player submenus hierarchy summary

Figure 54 hereafter yields a clearer view of the different audio player sub-menus hierarchy.

The “No Media” display shows up when the audio files or folders are not present on the SD card.

Figure 54. Audio player submenus hierarchy



4.4.5 Bird game

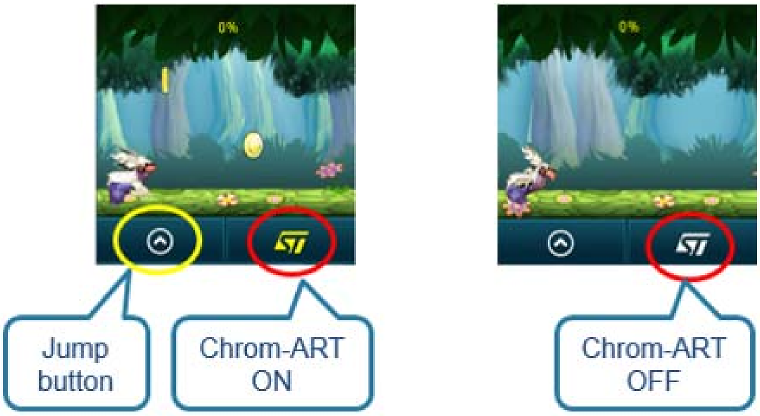
Overview

The Bird game is accessed thru a simple icon press as shown in *Figure 55*.

Figure 55. Starting Bird game

Playing the game means catching the golden coins while dodging the bullets at the same time. Pressing the left white arrow allows the bird to jump. Pressing the ST icon allows to enable / disable the Chrom-ART (DMA2D) hardware IP. The figure indicated at the top of the screen yields the CPU load in real time, highlighting the benefit of the Chrom-ART when enabled. The same is illustrated in *Figure 56*.

Figure 56. Playing Bird game



To exit the bird game, the user must slide the screen from up to bottom as shown in *Figure 57*.

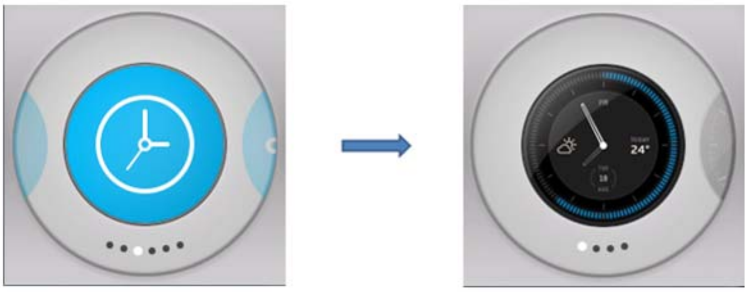
Figure 57. Exiting the Bird game

4.4.6 Watch

Overview

Similarly to the other menus, the watch menu is accessed thru a simple icon press as shown in *Figure 58*.

Figure 58. Starting watch menu



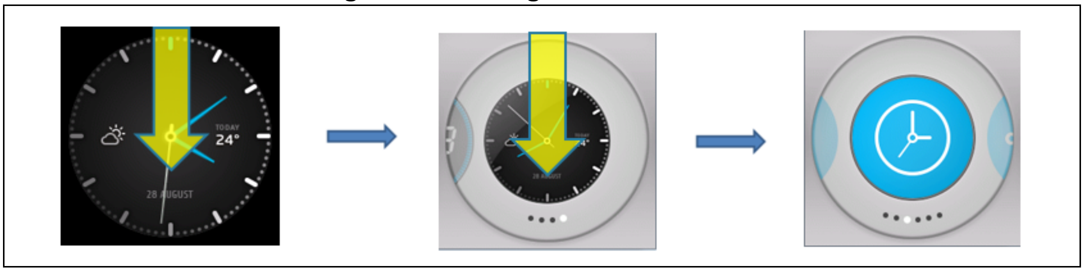
Several watch displays are proposed, each time illustrating a different watch type (analog or digital). The user can navigate from one type to another in sliding the screen left or right. Each display is frozen: touching the watch display allows to enter a new sub menu where the watch display is increased and the time updated on a second-basis (digits increasing or hands moving according to the clock type). *Figure 59* presents all the watch types as well as the sub-menu showing the clocks ticking.

Figure 59. Watch types

The difference between the Lite and Full TouchGFX demos lies in the number of watches types where only two are shown for the Lite demo.

To exit any watch sub-menu or the watch menu itself, the user must slide the screen from up to bottom as shown in *Figure 60*.

Figure 60. Exiting the watch menus



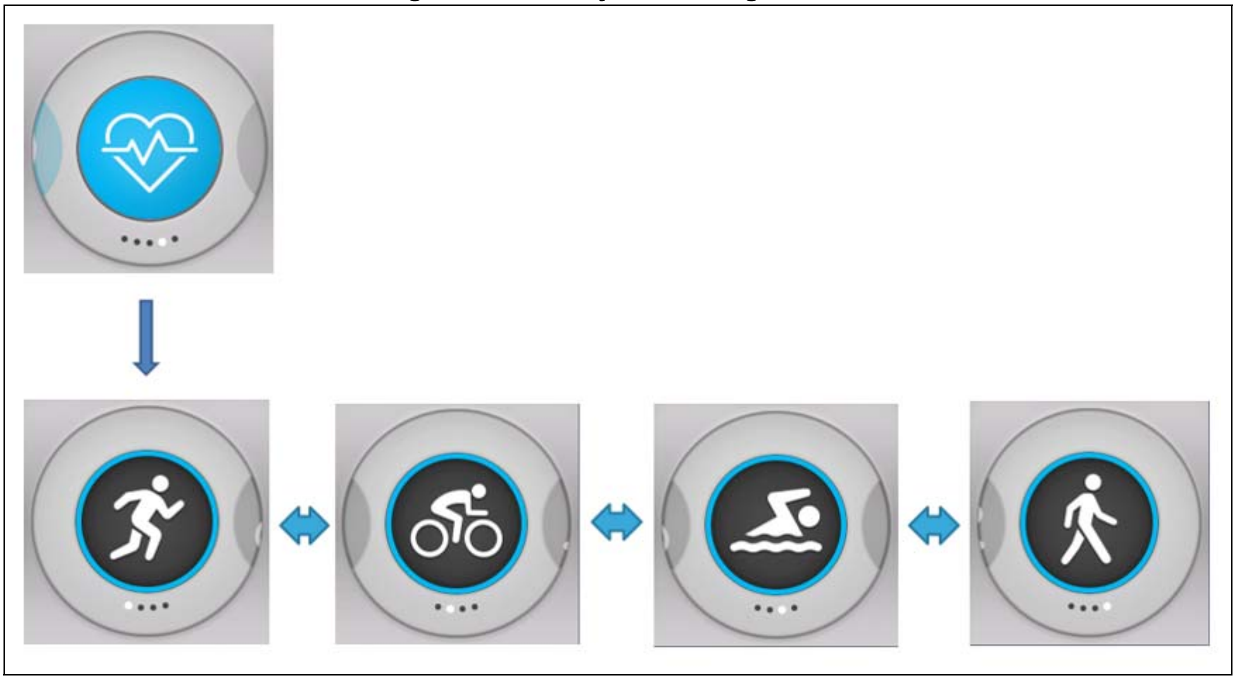
4.4.7 Activity monitoring

Overview

The activity monitoring menu relies on simulation to describe the different graphic displays that can be used to track running, cycling, swimming or walking activities.

Entering activity monitoring menu is achieved by a simple icon press as shown in *Figure 61*. Next, sliding left or right allows to navigate thru the different activities.

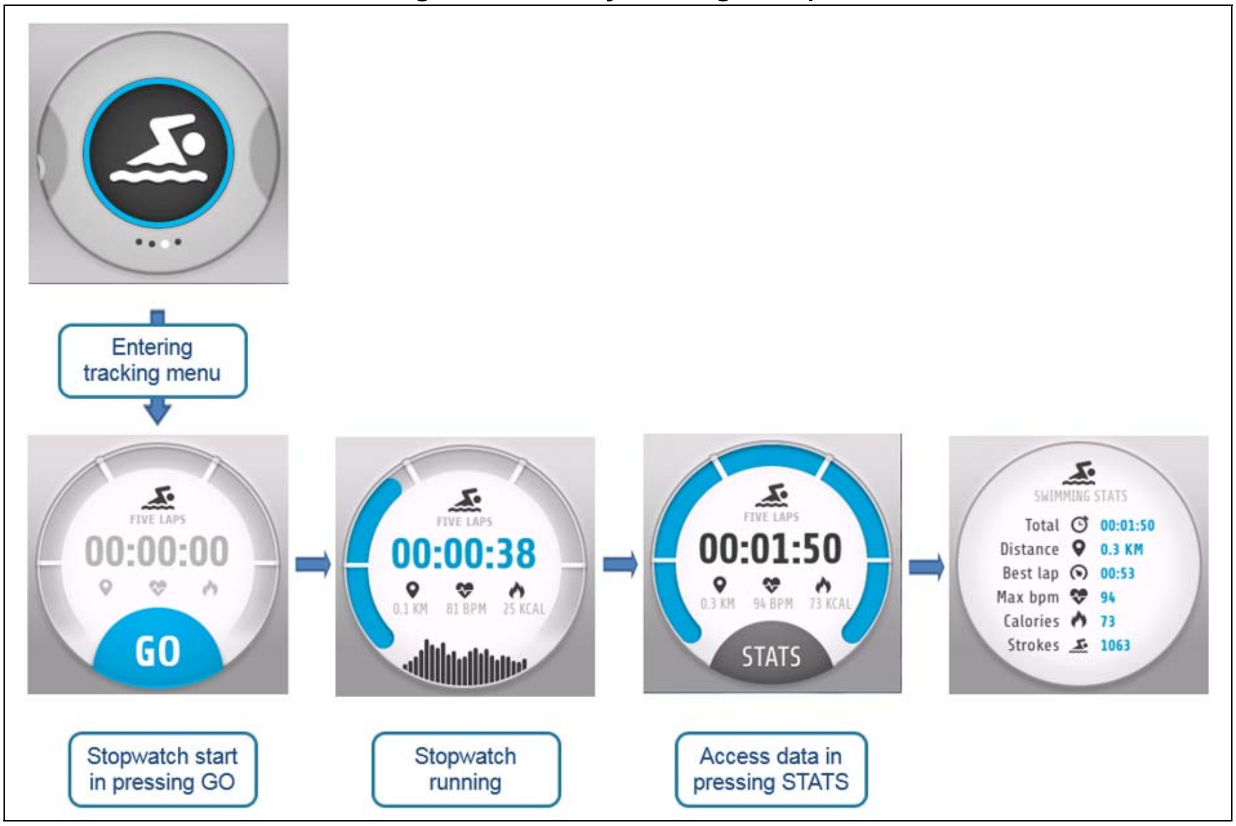
Figure 61. Activity monitoring menus



For each activity, pressing the relevant icon allows to enter the tracking menu. The first icon is that of a stopwatch that is started by a press on the GO button. The stopwatch runs while at the same time, distance, heart rate and calories counters are ticking.

When the simulation is ended, a press on the STATS button allows to retrieve the desired information. *Figure 62* describes these different displays in the swimming activity case

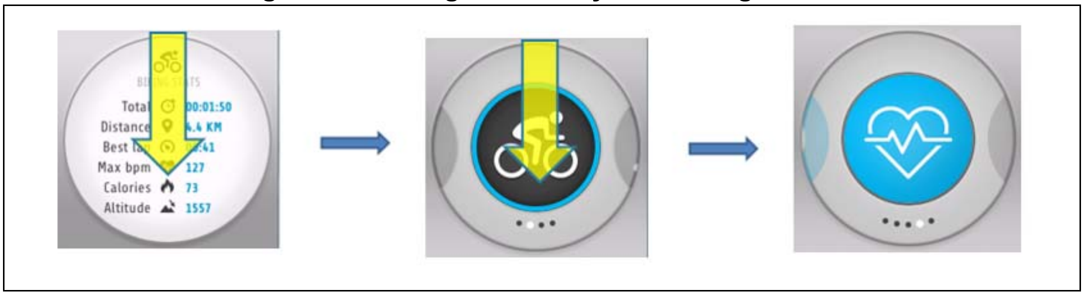
Figure 62. Activity tracking example



The difference between the Lite and Full TouchGFX demos lies in the number of activities monitoring: only two are available in the Lite demo (running and walking).

To exit any activity monitoring sub-menu or the activity monitoring menu itself, the user must slide the screen from up to bottom as shown in *Figure 63*.

Figure 63. Exiting the activity monitoring menus

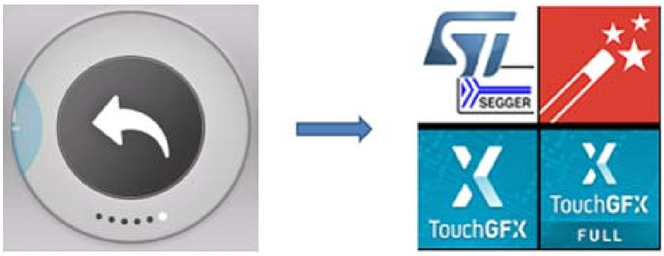


4.4.8 TouchGFX demo exit

Overview

The last icon in the TouchGFX demo menus sequence allows to come back to the main desktop as indicated in *Figure 64*. Pressing this icon triggers a software reset. It allows to quit the TouchGFX demo to choose to run the same or another one.

Figure 64. Exiting TouchGFX demo



5 Demonstration firmware settings

5.1 Clock control

The following clock configurations are used in the demonstration firmware:

- SYSCLK: 80MHz (PLL) from MSI 8MHz (RUN voltage range 1)

The following oscillators and PLL are used in the demonstration firmware:

- MSI (8 MHz) as PLL source clock
- LSI (32 KHz) as RTC clock source
- PLL main output at 80Mhz
- PLLSAI1 output at 48Mhz (PLL48M2CLK) for USB/SDMMC and configurable frequencies (PLLSAI1CLK) for SAI1:
  - PLLSAI1_VCO= 8 Mhz * PLLSAI1N = 8 * 24 = VCO_192M
  - SAI_CK_x = PLLSAI1_VCO/PLLSAI1P = 192/7 = 11.294 Mhz

5.2 Peripherals

The peripherals used in the demonstration firmware are listed in *Table 4*.

Table 4. Peripherals list

Used peripherals	Application/module
ADC	Idd application
CORTEX	NVIC services
DFSDM	Audio record application
DMA	Audio application and all applications with storage unit accesses on microSD card
DMA2D	Image transfer on LCD internal buffer
EXTI	Pushbutton and Idd application
FLASH	System settings
FMC	LCD interface
GPIO	All applications
I2C	IO expander usage on board
PWR	System and Idd application
QSPI	Graphics demos resources
RCC	System application and BSP drivers (SD/Audio)
RTC	System application and kernel backup service
SAI	Audio applications
SD	All applications with storage unit accesses
TIM	System temperature application (PWM)

5.3 Interrupts / Wakeup pins

The interrupts used in the demonstration firmware are listed *Table 5*.

Table 5. Interrupts list

Interrupts	Application/module	Priority, SubPriority (highest=0,0)
DMA1 Channel4	Audio record applications (DFSDM0 / left audio in)	5,0
DMA1 Channel5	Audio record applications (DFSDM0 / right audio in)	5,0
DMA2 Channel1	Audio player applications (SAI1 block A)	5,0
DMA2 Channel2	Audio player applications (SAI1 block B)	5,0
DMA2 Channel5	SD card data transfer in application with storage	6,0
EXTI Line 5	Idd application (wakeup Interrupt)	15,15
EXTI Line 8	SD card pin detection	5,0
EXTI Line 14	Pushbutton used for Idd application calibration	8,0
I2C2_EV_IRQn	I2C2 interrupt requests	0,0
I2C2_ER_IRQn	I2C2 errors	0,0
OTG_FS_IRQn	USB device application	7,0
RTC_WKUP_IRQn	LCD screen dimming applications	0,0
SAI1_IRQn	Audio applications	5,0
SDMMC1_IRQn	All applications with microSD storage	5,0
SysTick	CortexM4 system timer for OS tick	15, 0
TIM6_DAC_IRQn	Time base source	0,0

5.4 System memory configuration

The system memory areas used in the demonstration firmware are indicated *Table 6*.

Table 6. Memories areas

Memory	Start Address	Application
Internal Flash	0x80000000	Demonstration firmware run code and constants
Internal SRAM1	0x20000000	Demonstration firmware data (FreeRTOS heap included)
Internal SRAM2	0x10000000	Kernel memory area (64Kbytes) for kernel graphics and log management
External NOR	0x90000000	External QSPI memory flash where graphical resources are located (bitmaps)

5.5 Low power strategy

The STM32CubeL496G-Discovery firmware demonstration is designed to highlight the low power consumption capabilities of both the STM32L496AG MCU and the 32L496GDISCOVERY board.

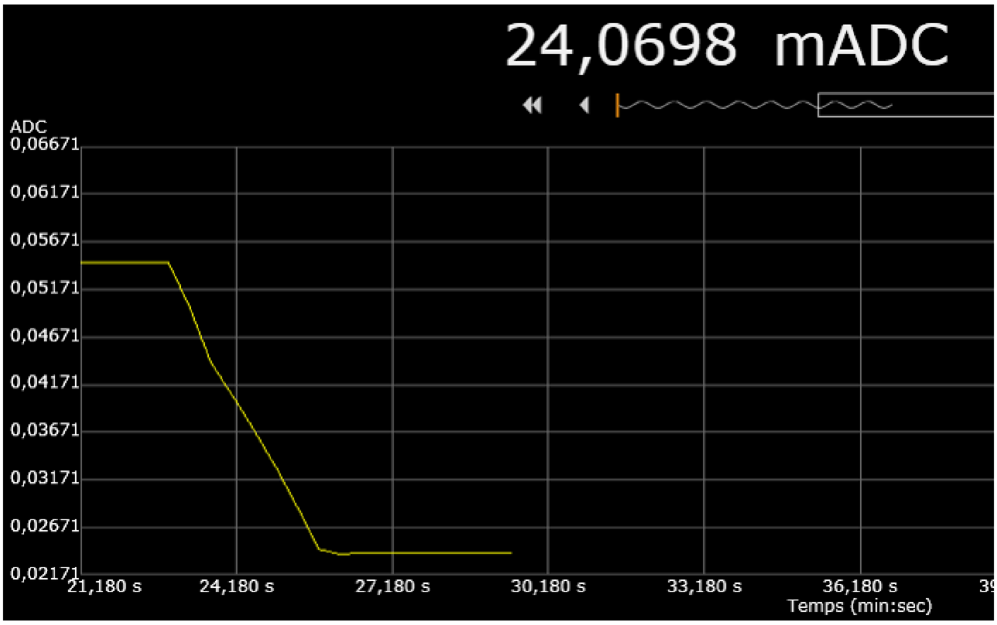
Screen dimming implemented in STemWin and TouchGFX demos illustrates the board power consumption gain when the LCD brightness is reduced. The backlight pin level is toggled on a PWM scheme based on TIM5 timer.

First, the timer output clock is set to 24 KHz and after a 7-sec period during which no activity is detected, the screen brightness is reduced from an activity level of 100 % to 5 % within two seconds. More precisely, when screen dimming is complete, the backlight level obeys the following pattern: off during 25.65 ms then on during 1.35 ms.

As soon as any activity is detected (e.g. touch screen press), dimming is immediately disabled and brightness activity level set back to 100%.

Figure 65 shows the board power consumption decrease when dimming is applied (activity level is gradually reduced over several examples in the figure). Y axis scale is in A.

Figure 65. Board power consumption decrease upon screen dimming enabling



STemWin demo carries on power consumption saving further: MCU power consumption is reduced in non-activity period in entering Low Power Sleep mode.

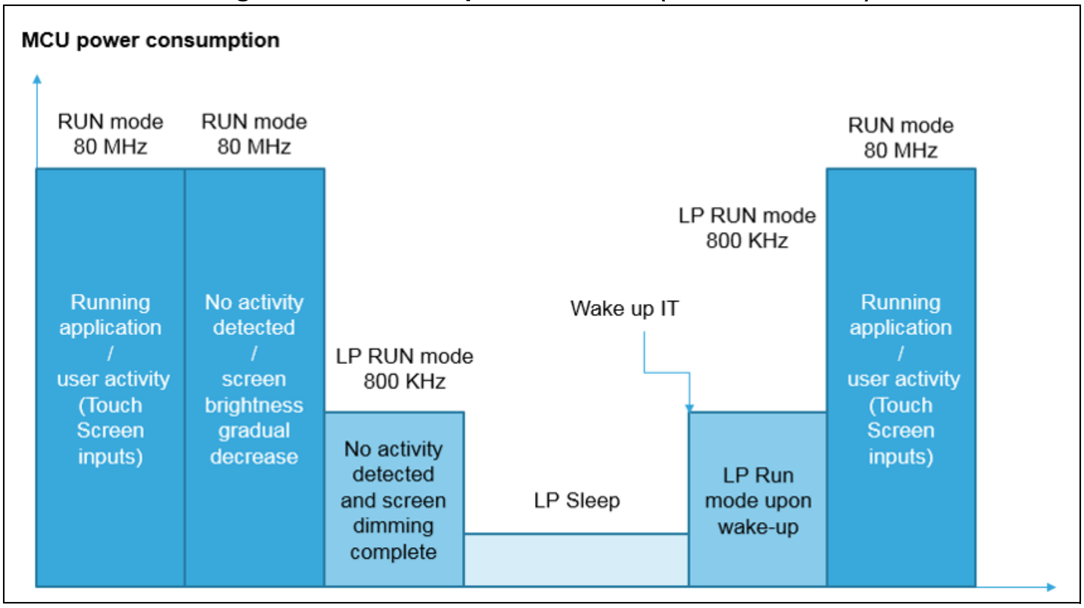
Once dimming is over, if no new activity is detected, the MCU system clock frequency is reduced to 800 KHz (i.e. MSI range 3).

Next, Low Power Run mode is entered, immediately followed by Low Power Sleep mode.

Any interruption allows to exit this LP Sleep: in that case, LP run mode is disabled (the MCU is in Low Power Run mode when exiting LP Sleep) and the system clock frequency is set back to 80 MHz.

Such scheme is described in *Figure 66*.

Figure 66. MCU low power scheme (STemWin demo)



Appendix A briefly describes the board needed hardware and software updates if the user wants to simultaneously achieve screen dimming and Stop2 low power mode for the MCU.

5.6 FreeRTOS resources

The next section only deals with STemWin demo since the source code of the latter is the only one provided with the L4 FW Cube package.

The 32L496GDISCOVERY firmware STemWin demonstration is designed on top of CMSIS-OS drivers based on FreeRTOS. Resources used in the firmware demonstration are listed hereafter.

As a reminder FreeRTOS configuration is described in *FreeRTOSConfig.h* file.

5.6.1 Tasks

Table 7. OS tasks

Task entry point	Description	Function (File)	Stack size (words)	Priority
MFX_Thread	Main application core (kernel)	k_MfxInit() k_mfx.c	2*128	osPriorityRealtime
osAudio_Thread	Audio Player and Recorder applications	AUDIOPLAYER_Init() (audio_player_app.c) AUDIO_RECORDER_Init() (audio_recorder_app.c)	4*128	osPriorityRealtime
GUIThread	GUI core (STemWin) Idle task (background)	main() (main.c)	32*128	osPriorityNormal

Table 7. OS tasks (continued)

Task entry point	Description	Function (File)	Stack size (words)	Priority
STORAGE_Thread	Storage management (kernel)	k_Storage_Init() (k_storage.c)	128	osPriorityLow
osRun_Mode_Thread	Infinite while(1) used for Run mode power consumption measurements	Idd_RunEnter() Idd_LprEnter() Iddmeasure.c	128	osPriorityNormal

5.6.2 Message Queues

Table 8. OS messages queues

QueueId	Description	Function (File)	Queue depth (word)
StorageEvent	Queue to receive storage event	K_StorageInit() (k_storage.c)	10
AudioEvent	Audio player and Audio Recorder input event	AUDIOPLAYER_Init() (Audioplayer.c)	1
MfxEvent	MFX event	k_MfxInit() k_mfx.c	3

5.6.3 Mutex

A mutex *MfxIddSemaphore* is defined to control the main application entry in low power mode by first insuring the board components with their respective IOs are in low power mode and then setting the MCU is low power consumption.

The mutex *MfxIddSemaphore* is released upon wakeup from an EXTI lines associated to joystick buttons.

5.6.4 Heap

The FreeRTOS heap size is defined in FreeRTOSConfig.h as follows

```
95 #define configTOTAL_HEAP_SIZE                ( ( size_t ) ( 40 * 1024 ) )
```

Heap usage in the firmware demonstration is dedicated to:

- OS resources (Tasks, Queues, Mutexes, Memory allocation)
- Application memory allocations requirements

Table 9. heap usage

Applications	Description	Function (File)	Memory requirements (bytes)
USB Device	Mass storage class handle	USBD_MSC_Init() (usbd_msc.c)	< 10 Kbytes
Audio Record	Record buffer	AudioRecorder_Start() (Audiorecorder.c)	2048

5.7 Programming firmware application

First of all install the ST-LINK/V2.1 driver available on ST website.

There are two ways of programming the STM32L496G-EVAL board.

5.7.1 Using Binary file

Upload the binary STM32CubeDemo_STM32L496G-Discovery-VX.Y.Z.hex from the firmware package available under Projects\STM32L496G-Discovery\Demonstrations\Binary using your preferred in-system programming tool.

5.7.2 Using preconfigured projects

Choose one of the supported tool chains and follow the steps below:

- Open the application folder: Projects\STM32L496G-Discovery\Demonstrations
- Chose the desired IDE project (EWARM for IAR, MDK-ARM for Keil, SW4STM32)
- Double click on the project file (for example Project.eww for EWARM)
- Rebuild all files: Go to Project and select Rebuild all
- Load the project image: Go to Project and select Debug
- Run the program: Go to Debug and select Go

6 Demonstration firmware footprints

As stated in *Section 4.1*, four different demonstrations are provided by the demonstration firmware:

- STemWin graphic demo
- Embedded Wizard graphic demo
- TouchGFX graphic demo (Lite)
- TouchGFX Full graphic demo

Table 10 provides the footprints of each demo (Flash size for the binaries and Quad-SPI size for the graphic resources).

Table 10. Demonstrations footprints

Graphic demonstration	Requested size in Flash (in KByte)	Requested size in QSPI (in MByte)
STemWin	359.2	1.17
Embedded Wizard	134.7	2.94
TouchGFX (lite)	202.2	3.30
TouchGFX (full)	221.4	6.09
32L496GDISCOVERY available size	1024	8

These figures explain why it is possible to fit STemWin, TouchGFX Lite and Embedded Wizard together with the 8-Mbyte large Quad-SPI external flash but how fitting STemWin, TouchGFX Full and Embedded Wizard at the same time is not possible.

7 Kernel description

7.1 Overview

The role of the demonstration kernel is mainly to provide a generic platform that controls and monitors all the application processes with minimum memory consumption. The kernel provides a set of friendly services that simplify module implementation by allowing access to all the hardware and firmware resources through the following tasks and services:

- Hardware and modules initialization:
  - BSP initialization (LEDs, Touchscreen, LCD, RTC, Audio, MFX, SRAM and QSPI)
- Graphical and main menu management.
- Memory management
- Storage management (microSD card)
- System monitoring and settings
- CPU utilities (CPU usage, running tasks)

7.2 Kernel core files

Table 11. Kernel core files

Function	Description
main.c	Main program file
stm32l4xx_it.c	Interrupt handlers for the application
k_calibration.c	Touchscreen kernel calibration manager
k_menu.c	Kernel menu and desktop manager
k_module.c	Module manager
k_rtc.c	Real-time clock manager
k_startup.c	Demonstration startup windowing process
k_storage.c	Storage manager
startup_stm32l496xx.s	Startup file

7.3 Kernel initialization

The first task of the kernel is to initialize the hardware and firmware resources to make them available to its internal processes and the modules around it. The kernel starts by initializing the HAL, system clocks and then the hardware resources needed during the middleware components:

- LEDs
- Push-button,
- Touchscreen
- IO expanders
- SRAM
- External QSPI flash
- RTC

Once the low level resources are initialized, the kernel performs the STemWin GUI library initialization and prepares the following common services:

- Memory manager
- Storage unit,
- Modules manage
- Kernel log

Upon full initialization phase, the kernel adds and links the system and user modules to the demonstration core.

7.4 Kernel processes and tasks

The kernel is composed of a main task managed by FreeRTOS through the CMSIS-OS wrapping layer:

- *GUI Thread*: once the demonstration is initialized by the main function *main( )*, this task handles the graphical background task when requested by the STemWin.

```
/**
  * @brief  Start task
  * @param  argument: pointer that is passed to the thread function as start argument.
  * @retval None
  */
static void GUIThread(void const * argument)
{
  if(TouchScreen_IsCalibrationDone() == 0)
  {
    Touchscreen_Calibration();
  }

  /* Enable TS interrupt */
  if (BSP_TS_ITConfig() != TS_OK) {
    Error_Handler();
  }

  if (software_reset_flag == 0)
  {
    /* Demo Startup */
    k_StartUp();
  }

  /* Initialize Storage Units */
  k_StorageInit();

  /* Show the main menu */
  k_InitMenu();

  /* Gui background Task */
  while(1) {
    GUI_Exec(); /* Do the background work ... Update windows etc.) */
```

7.5 Kernel graphical aspect

This section emphasizes aspects applicable only to the STemWin demo (since STemWin demo is the only software is provided in the Cube L4 FW package).

This demonstration is built around the STemWin Graphical Library, based on SEGGER emWin one. STemWin is a professional graphical stack library, enabling Graphical User Interfaces (GUI) building up with any STM32, any LCD and any LCD controller, taking benefit from STM32 hardware accelerations whenever possible. In STM32L496G-Disco, Chrom-ART (DMA2D) IP is the IP allowing hardware acceleration.

The graphical aspect of the STM32Cube demonstration is divided into two main graphical components:

- The startup window: showing the progress of the hardware and software initialization (see *Figure 67*).

Figure 67. Start-up window

- The main desktop that handle the main demonstration menu and the numerous kernel and modules control, once the user has picked up STemWin demonstration (see *Figure 68*).

Figure 68. Main desktop

7.6 Kernel menu management

The main demonstration menu is initialized and launched by the GUI thread. Before the initialization of the menu the following actions are performed:

- Draw the background image
- Restore general settings from backup memory.
- Setup the main desktop callback to manage main window messages.

The icon view widget: contains the icons associated to added modules. User can launch a module by a simple click on the module icon.

A module is launched on simple click on the associated icon by calling to the startup function in the module structure; this is done when a WM_NOTIFICATION_RELEASED message arrives to the desktop callback with ID_ICONVIEW_MENU:

```
/**
  * @brief  Callback routine of desktop window.
  * @param  pMsg: pointer to data structure of type WM_MESSAGE
  * @retval None
  */
static void _cbBk(WM_MESSAGE * pMsg) {

  (...)

  switch (pMsg->MsgId)
  {
  case WM_NOTIFY_PARENT:
    Id     = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;

    switch (NCode)
    {
    case WM_NOTIFICATION_RELEASED:
      if (Id == ID_ICONVIEW_MENU)
      {

        if(sel < k_ModuleGetNumber())
        {
          module_prop[sel].module->startup(pMsg->hWin, 0, 26);
          sel = 0;
        }
      }
      else if (Id == ID_BUTTON_BKGND)
      {
        /* Create popup menu after touching the display */
        _OpenPopup(WM_HBKWIN, _aMenuItems, GUI_COUNTOF(_aMenuItems),0 , 25);
      }
      break;
```

7.7 Modules manager

The main demonstration menu is initialized by the *main()* function.

The modules are managed by the kernel: it is responsible of initializing the modules, the hardware and GUI resources relative to the modules and the common resources such as the storage Unit, the graphical widgets and the system menu.

Each module provides the following functionalities and properties:

1. Icon and graphical component structure.
2. Method to startup the module.
3. Method to close down safety the module (example; Hot unplug for MS flash disk)
4. Method to manage low power mode (optional)
5. The Application task
6. The module background process (optional)
7. Specific configuration
8. Error management

The modules could be added in run time to the demonstration and can use the common kernel resources. The following code shows how to add a module to the demonstration:

```
/* Add Modules*/
k_ModuleInit();

k_ModuleAdd(&STemWin_board);
k_ModuleAdd(&EW_board);
k_ModuleAdd(&TGFX_board);
k_ModuleAdd(&TGFX_FULL_board);


k_ModuleAdd(&video_player_board);
k_ModuleAdd(&audio_player_board);
k_ModuleAdd(&audio_recorder_board);
k_ModuleAdd(&INFORMATION_board);

k_ModuleAdd(&USB_Storage_board);
k_ModuleAdd(&idd_measure_board);
k_ModuleAdd(&analog_clock_board);
k_ModuleAdd(&return_board);
```

A module is a set of function and data structures that are defined in a data structure that provides all the information and pointers to specific methods and functions to the kernel. The latter checks the integrity and the validity of the module and inserts its structure into a module table. Each module is identified by a unique ID. When two modules have the same ID, the Kernel rejects the second one. The module structure is defined as follows:

```
typedef struct
{
 uint8_t      id;
 const char   *name;
 GUI_CONST_STORAGE GUI_BITMAP  *icon;
 void         (*startup) (WM_HWIN , uint16_t, uint16_t );
 void         (*DirectOpen) (char * );
}
K_ModuleItem_Typedef;
```

- id: unique module identifier.
- name: pointer to module name
- icon: pointer to module icon frame (bitmap in array)
- Startup: the function that create the module frame and control buttons
- DirectOpen: the function that creates the module frame and launches the media associated to the file name selected in the file browser linked to a specific file extension.

7.8 Backup and settings configuration

The 32L496GDISCOVERY firmware demonstration saves several kinds of information in the RTC backup registers (32 bits data width).

RTC_BKP_DR30	When the user picks up Embedded Wizard or TouchGFX demo, the software resets to restart and jump at the address where the picked up demo binary is stored. This address is stored in RTC_BKP_DR30.

Register is used in the following example:

```
/* Jump to new Sub Demo if no errors */
if(result == FR_OK)
{
  k_JumpToSubDemo(RTC->BKP30R, TARA_DEMO_FLASH_ADDRESS);
}
```

RTC_BKP_DR31	Since no flag is available to indicate at software start that the system is resuming from shutdown mode, RTC_BKP_DR31 is used to record this information before entering shutdown mode. At software start, the back-up register is read to check whether or not coming back from shutdown.

Code below shows the check at startup:

```
/* Check if the system was resumed from shutdown mode,
   resort to RTC back-up register RTC_BKP31R to verify
   whether or not shutdown entry flag was set by software
   before entering shutdown mode.  */
if (READ_REG(RTC->BKP31R) == 1)
{
  WakeUpFromShutdown = SET;
}
else
{
  WakeUpFromShutdown = RESET;
}
```

RTC_BKP_DR27	Since software resets occurs to quit any sub-demo (STemWin, Embedded Wizard, TouchGFX), the ST logo must be shown only after a hardware reset. RTC_BKP_DR27 is used to detect a software reset.

Code below illustrates the use of RTC_BKP_DR27.

```
/* Check whether or not coming back from a software reset.
   Used to avoid displaying ST logo again */
software_reset_flag = READ_REG(RTC->BKP27R);
WRITE_REG(RTC->BKP27R, 1);
```

Touch Screen calibration parameters are saved by default in RTC -> BKP0R and RTC -> BKP1R to be easily retrieved if no calibration is requested by the user.

RTC_BKP_DR0	First Touch Screen calibration parameter
RTC_BKP_DR1	Second Touch Screen calibration parameter

Calibration parameters are saved as well in FLASH to make sure the parameters are saved even after a power off. Three 64-bit long words are used to store three 32-bit long parameters.

```
/* Start calibration_data region definition to map calibration_data[] array:
   calibration_data[] stores the calibration parameters in Flash memory */
#pragma section =".calibration_data"
#pragma default_variable_attributes = @ ".calibration_data"
uint32_t calibration_data[3*2];
/* Three 64-bit long words used to store 3 32-bit long parameters
  calibration_data[0] = 0 / 1 : very first calibration: done no / yes
  calibration_data[2] = data1.d32 calibration parameter
  calibration_data[4] = data2.d32 calibration parameter */

/* Stop placing data in section calibration_data region */
#pragma default_variable_attributes =
```

where *calibration_data* is defined in the scatter file as shown in the IAR scatter file below.

```
define symbol __ICFEDIT_calibration_data_start__   = 0x080FFE00;
define symbol __ICFEDIT_calibration_data_end__     = 0x080FFFFF;
```

Software hereafter describes how the parameters are saved in Flash.

```
/* Save calibration parameters in Flash */
__HAL_FLASH_CLEAR_FLAG(FLASH_FLAG_OPTVERR);
HAL_FLASH_Unlock();
__HAL_FLASH_CLEAR_FLAG(FLASH_FLAG_ALL_ERRORS);

/* First, erase Flash to be able to write afterwards */
/* Fill EraseInit structure*/
EraseInitStruct.TypeErase   = FLASH_TYPEERASE_PAGES;
EraseInitStruct.Banks       = CALIBRATION_PARAM_FLASH_BANK;
EraseInitStruct.Page        = CALIBRATION_PARAM_FLASH_PAGE;
EraseInitStruct.NbPages     = CALIBRATION_PARAM_FLASH_NB_PAGES;

if (HAL_FLASHEx_Erase(&EraseInitStruct, &PAGEError) != HAL_OK)
{
  Flash_error = HAL_FLASH_GetError();
  BSP_LCD_DisplayStringAt(0, 240 - 65, (uint8_t *)"Calibration parameters erasing error", CENTER_MODE);
}

/* Write parameters */
   SET_BIT(FLASH->CR, FLASH_CR_PG);
  __HAL_FLASH_CLEAR_FLAG(FLASH_FLAG_ALL_ERRORS);
/* Save calibration 'done' information */
if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, (uint32_t)&calibration_data, (uint64_t) 1) != HAL_OK)
{
  Flash_error = HAL_FLASH_GetError();
  BSP_LCD_DisplayStringAt(0, 240 - 65, (uint8_t *)"Calibration parameters saving error", CENTER_MODE);
}
/* Save data1.d32 calibration parameter */
if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, (uint32_t)&calibration_data[2], (uint64_t) (data1.d32)) != HAL_
{
  Flash_error = HAL_FLASH_GetError();
  BSP_LCD_DisplayStringAt(0, 240 - 65, (uint8_t *)"Calibration parameters saving error", CENTER_MODE);
}
/* Save data2.d32 calibration parameter */
if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, (uint32_t)&calibration_data[4], (uint64_t) (data2.d32)) != HAL_
{
  Flash_error = HAL_FLASH_GetError();
  BSP_LCD_DisplayStringAt(0, 240 - 65, (uint8_t *)"Calibration parameters saving error", CENTER_MODE);
}
```

7.9 Storage units

The STM32Cube demonstration resorts to a storage unit that is used to retrieve video and audio media as well as Embedded Wizard and TouchGFX binaries and graphic resources.

The storage unit is initialized during the platform startup and thus it is available to all the modules during the STM32Cube Demonstration run time provided the media component is present (Unit 0: microSD card).

The unit is accessible through the standard I/O operations offered by the FatFS used in the development platform. The microSD card flash unit is identified as the Unit 0 and available only if a microSD card disk flash is inserted on the CN17 connector. The Unit is mounted automatically when the physical media is connected. The implemented functions in the file system interface to deal with the physical storage unit are listed in *Table 12*.

Table 12. File system interface functions

Function	Description
disk_initialize	Initialize disk drive
disk_read	Interface function for a logical page read
disk_write	Interface function for a logical page write
disk_status	Interface function for testing if unit is ready
disk_ioctl	Control device dependent features

The full APIs functions set given by the file system interface are listed in *Table 13*.

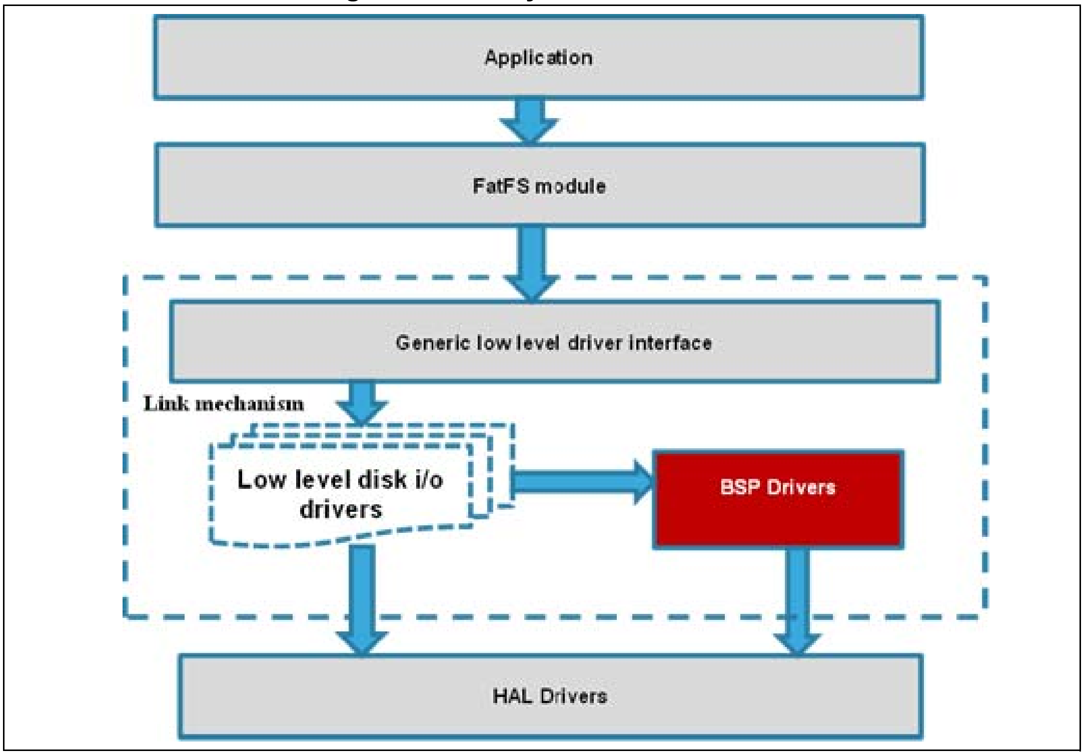
Table 13. File system APIs

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_synv	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
f_putc	Write a character
f_puts	Write a string
f_printf	Write a formatted string

For the FAT FS file system, the page size is fixed to 512 bytes.

The software architecture is described in *Figure 69*.

Figure 69. File System architecture



8 How to create a new module

A module is composed of two main parts:

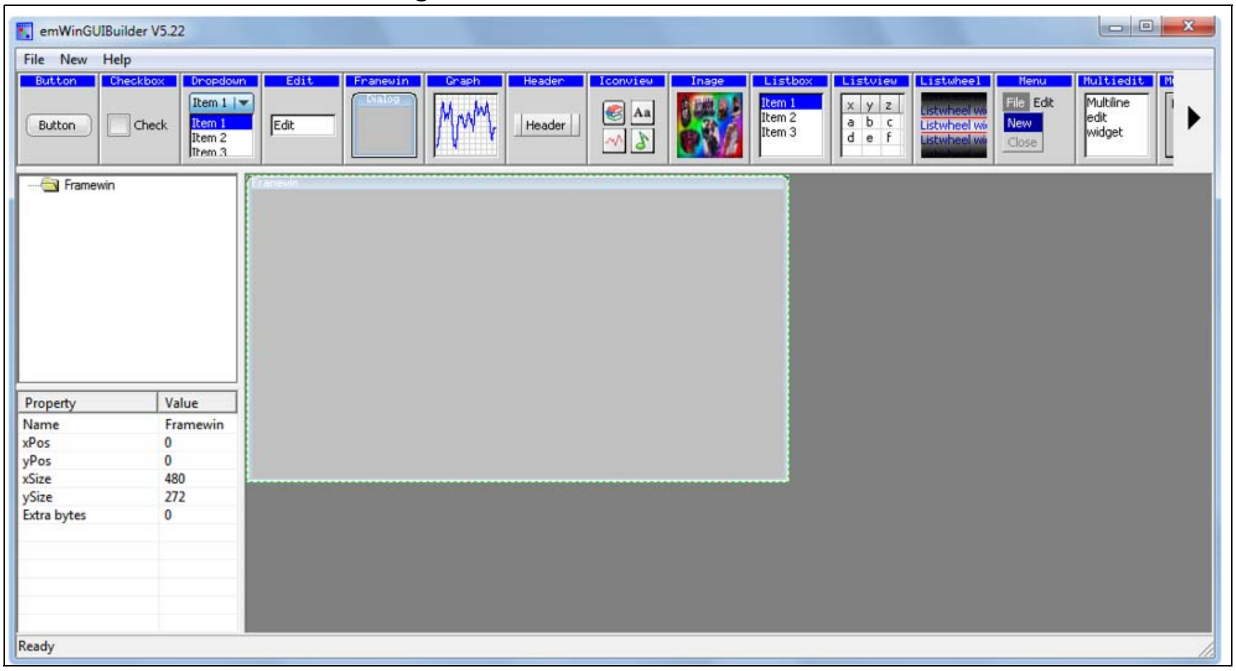
- Graphical aspect : the main window frame and module's controls
- Functionalities : module functions and internal processes

8.1 Creating the graphical aspect

The graphical aspect consists of the main frame window in addition to the set of the visual elements and controls (buttons, checkboxes, progress bars…) used to control and monitor the module's functionalities.

The STM32Cube demonstration package provides a PC tool; the GUIBuilder (see *Figure 70*) that allows easily and quickly creating the module frame window and all its components in few steps. For more information about the GUI Builder, refer to the emwin User and reference guide (UM03001).

Figure 70. The GUI Builder overview



The GUI Builder only takes few minutes to totally design the module appearances using "drag and drop" commands and then generate the source code file to be included into the application.

The file generated is composed of the following main parts:

- A resource table: it's a table of type GUI_WIDGET_CREATE_INFO, which specifies all the widgets to be included in the dialog and also their respective positions and sizes.
- A dialog callback routine: described more in detail in *Section 4.3* (it is referred to as "main module callback routine").

8.2 Graphics customization

After the basic module graphical appearance is created, it is then possible to customize some graphical elements, such as the buttons, by replacing the standard aspect by the user defined image. To do this, a new element drawing callback should be created and used instead of the original one.

Below is an example of a custom callback for the Play button:

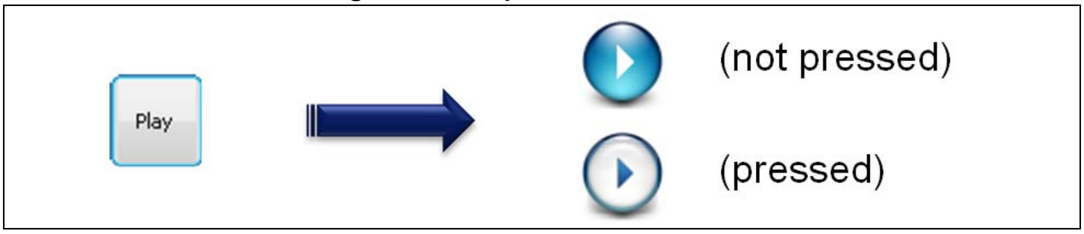
```
/**
  * @brief  callback for play button
  * @param  pMsg: pointer to data structure of type WM_MESSAGE
  * @retval None
  */
static void _cbButton_play(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
    case WM_PAINT:
      _OnPaint_play(pMsg->hWin);
      break;
    default:
      /* The original callback */
      BUTTON_Callback(pMsg);
      break;
  }
}
```

On the code portion above, the _OnPaint_play routine contains just the new button drawing command.

Note that the new callback should be associated to the graphical element at the moment of its creation, as shown below:

```
hItem = BUTTON_CreateEx(148,140,50,50,pMsg->hWin,WM_CF_SHOW,0,ID_PLAY_BUTTON);
WM_SetCallback(hItem,_cbButton_play);
```

Figure 71. Graphics customization



8.3 Module implementation

Once the graphical part of the module is finalized, the module functionalities and processes can be added. It begins with the creation of the main module structure as defined in *Section 7.1*. Then, each module has its own Startup function which simply consists of the graphical module creation, initialization and link to the main callback:

```
/**
  * @brief  Module window Startup
  * @param  hWin: pointer to the parent handle.
  * @param  xpos: X position
  * @param  ypos: Y position
  * @retval None
  */
static void Startup(WM_HWIN hWin, uint16_t xpos, uint16_t ypos)
{
  GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), _cbDialog, hWin, xpos, ypos);
}
```

In the example above cbDialog refers to the main module callback routine. Its general skeleton is structured like the following:

```
/**
  * @brief  Callback routine of the dialog
  * @param  pMsg: pointer to data structure of type WM_MESSAGE
  * @retval None
  */
static void _cbDialog(WM_MESSAGE * pMsg){
  switch (pMsg->MsgId){
  case WM_INIT_DIALOG:
    /* Initialize graphical elements and restore backup parameters if any */
  case WM_NOTIFY_PARENT:
    Id     = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;
    switch(Id)  {
    case ID_BUTTON:
      switch (NCode)  {
      case WM_NOTIFICATION_RELEASED:
        /* Operation associated to the button */
      }
      (...)
```

The list of windows messages presented in the code portion above (WM_INIT_DIALOG and WM_NOTIFY_PARENT) is not exhaustive, but it represents the essential message IDs used:

- WM_INIT_DIALOG: allows initializing the graphical elements with their respective initial values. It is also possible here to restore the backup parameters (if any) that will be used during the dialog procedure.
- WM_NOTIFY_PARENT: describes the dialog procedure, for example: define the behavior of each button.

The full list of window messages can be found in the WM.h file.

8.4 Adding a module to the main desktop

Once the module appearance and functionality are defined and created, it still needs to be added to the main desktop view. This is done by adding it to the list (structure) of menu items: module_prop[ ], defined into k_module.h.

To do this, k_ModuleAdd() function should be called just after the module initialization into the main.c file.

Note that the maximum modules number in the demonstration package is limited to 15; this value can be changed by updating MAX_MODULES_NUM defined into k_module.c.

8.5 Module's direct open

If there is a need to launch the module directly from the file browser contextual menu, an additional method should be added in the module structure for the direct open feature. This callback is often named *_ModuleName_DirectOpen*.

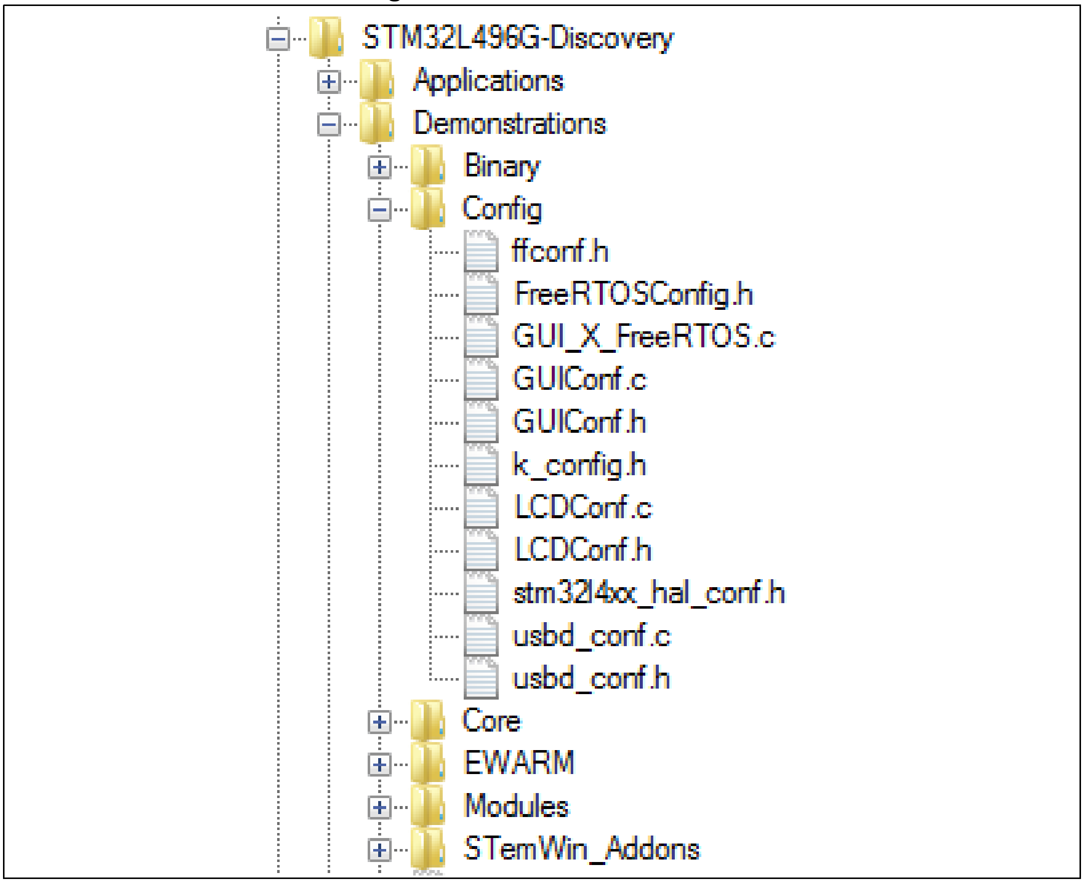
9 Demonstration customization and configuration

9.1 LCD configuration

The LCD is configured through the LCDConf.c file (*Figure 72*). The main configuration items are listed below:

- Multiple layers:
  - The number of layers to be used defined using GUI_NUM_LAYERS.
- Multiple buffering:
  - If NUM_BUFFERS is set to a value "n" greater than 1, it means that "n" frame buffers will be used for drawing operation. STM32L496-Discovery STemWin demonstration GUI_NUM_LAYERS = 2.

Figure 72. LCDConf location



9.2 Touchscreen calibration

This section provides additional explanations to *Section 4.1.2*.

- When the demonstration is launched for the first time, or

- when the discovery board is powered on and the JOY_SEL button pressed at the same time,

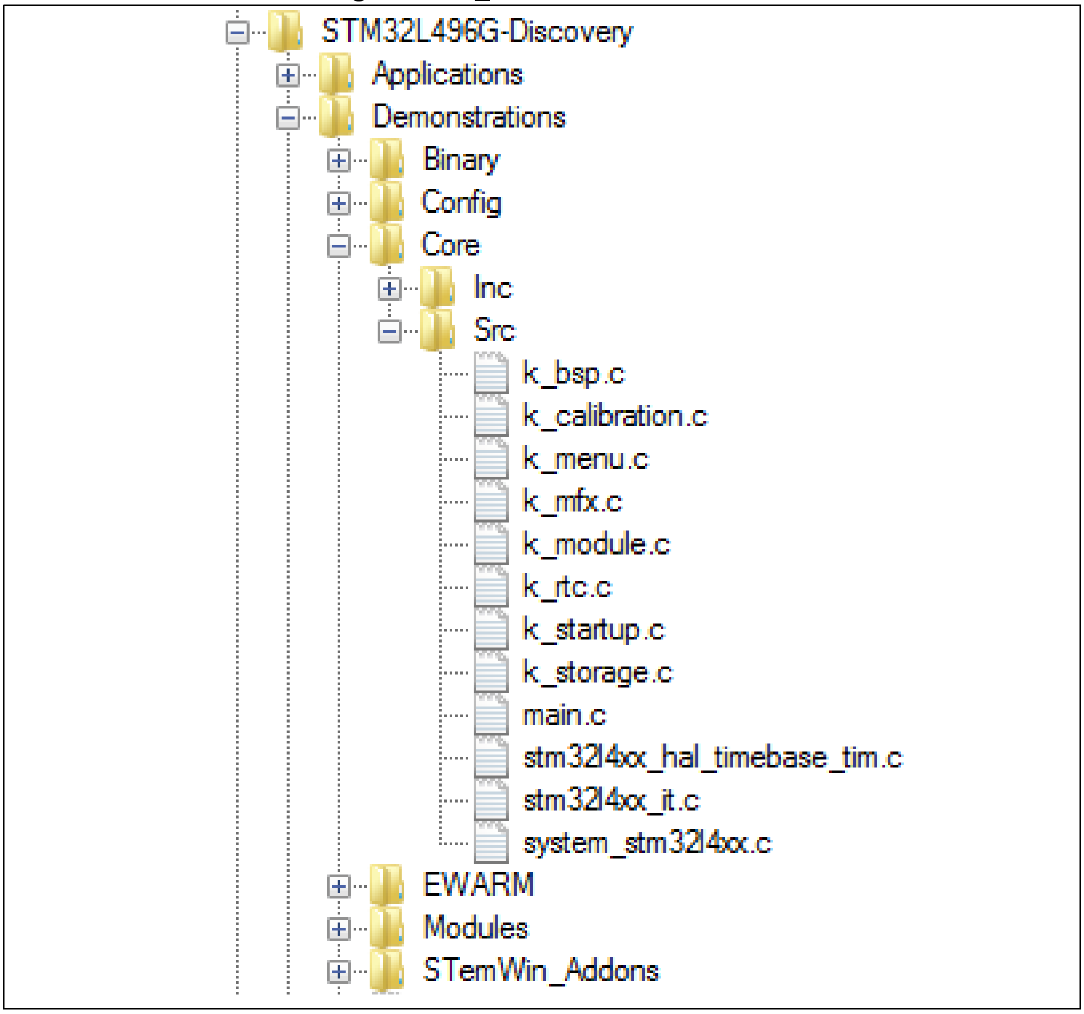
or

- when a software reset is carried out (e.g. with the RESET button press) and the JOY_SEL button pressed at the same time,

then the touchscreen calibration is started.

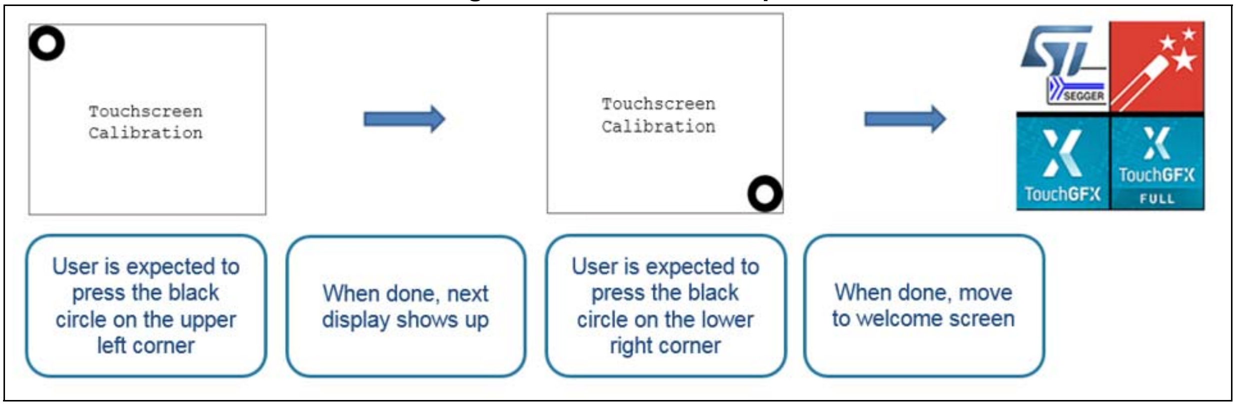
A full set of dedicated routines is included in the demonstration package and regrouped into k_calibration.c file shown in *Figure 73*.

Figure 73. k_calibration.c location



After the calibration screen is displayed, the user has to follow the displayed calibration instructions by touching the screen at the indicated positions (*Figure 74*). This allows to get the physical Touch screen values that will be used to calibrate the screen.

Figure 74. Calibration steps



Once this runtime calibration is done, as explained in *Section 7.8*, the touch screen calibration parameters are saved in RTC Backup data registers: RTC_BKP_DR0 and RTC_BKP_DR1 as well in FLASH at address 0x080FFE00.

At the next software or hardware reset, the parameters are automatically restored unless the user is pressing the joystick SEL button. In that case, the calibration process restarts.

Appendix A Simultaneous screen dimming and Stop 2 low power mode

STemWin demonstration sets the MCU in LP Sleep mode while the screen is dimmed. LP Sleep mode ensures the TIM5 timer clock is still running to guarantee screen dimming.

With such a setting, moving in Stop2 mode stops TIM5 timer clock: dimming is interrupted and screen is turned off.

Therefore, to reach a lower MCU power consumption and at the same time, keep the screen on and dimmed, the user must resort to the LPTIM1 Low Power timer that is still running when the MCU is in Stop2 mode.

Using LPTIM1 instead of TIM5 requires a slight hardware update which consists in connecting PI0 (PI0 at pin 1 of R77) and PG15 (at pin 1 of CN10) as shown in *Figure 75*.

User needs as well to:

- Update the BSP functions implementing the dimming feature to use LPTIM1 and not TIM5 (update BSP_LCD_ScreenDimmingOn() in stm32l496g_discovery_lcd.c, fill up HAL_LPTIM_MspInit())
- Drive the backlight thru PG15 instead of PI0. This means changing the following defines in stm32l496g_discovery_lcd.h

```
/* Backlight control pin */
#define LCD_BL_CTRL_PIN                     GPIO_PIN_0
#define LCD_BL_CTRL_GPIO_PORT               GPIOI
#define LCD_BL_CTRL_GPIO_CLK_ENABLE()       __HAL_RCC_GPIOI_CLK_ENABLE()
#define LCD_BL_CTRL_GPIO_CLK_DISABLE()      __HAL_RCC_GPIOI_CLK_DISABLE()
```

Figure 75. Driving backlight pin level thru LP Timer output

Revision history

Table 14. Document revision history

Date	Revision	Changes
23-Mar-2017	1	Initial release