UM2222 User manual

STM32CubeH7 demonstration platform

Introduction

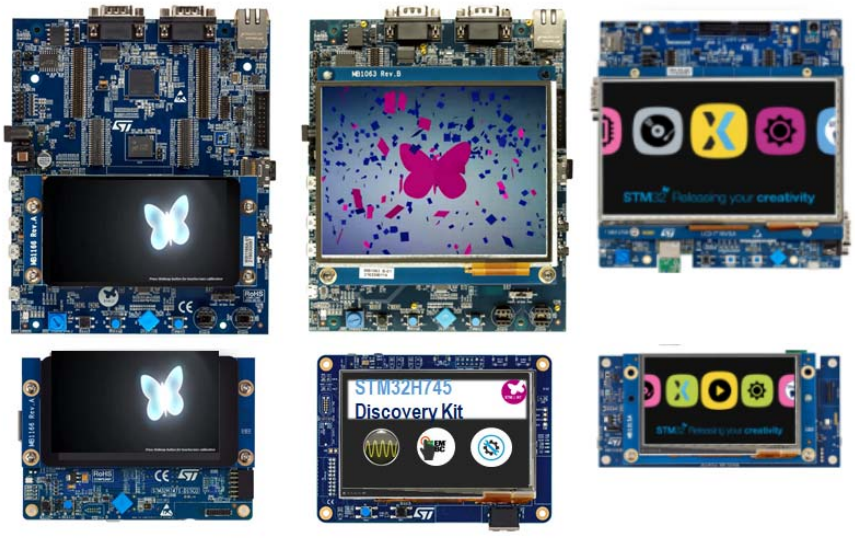
STM32Cube is an STMicroelectronics original initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

The STM32CubeH7 demonstration platform complements STM32Cube as a firmware package that offers a full set of software components based on a modular architecture, separately reusable in standalone applications. The STM32CubeH7 demonstration kernel manages all these modules, allowing the dynamic addition of new modules, and access to common resources (storage, graphical components and widgets, memory management, real-time operating system).

The STM32CubeH7 demonstration platform is built around the powerful STemWin graphical library and the FreeRTOS™ real-time operating system and uses almost the whole STM32 capability to offer a large scope of usage based on the STM32Cube HAL BSP and several middleware components.

The architecture uses the STM32CubeH7 demonstration core to make an independent central component that is usable with several RTOSs and third party firmware libraries through dedicated abstraction layers inserted between the STM32CubeH7 demonstration core and the associated modules and libraries.

The STM32CubeH7 demonstration supports STM32H7 Series devices and runs on STM32H743I-EVAL, STM32H745I-DISCO, STM32H747I-EVAL, STM32H747I-DISCO, STM32H747I-DISC1, STM32H7B3I-EVAL, and STM32H7B3I-DK boards.



Pictures are not contractual.

Contents

List of tables

List of figures

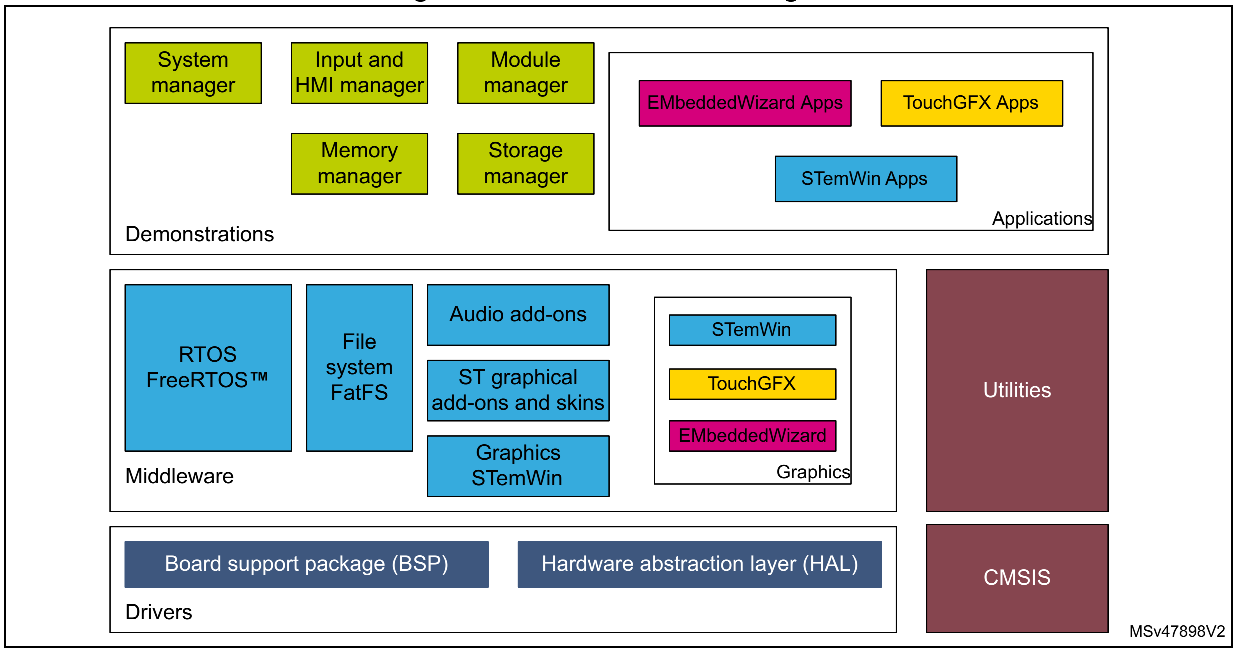
1 STM32Cube overview

STM32Cube is an STMicroelectronics original initiative to significantly improve designer's productivity by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:

- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
- STM32Cube MCU and MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeH7 for the STM32H7 Series microcontroller), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the portfolio of STM32 32-bit Arm®[a] Cortex®-based microcontrollers
  - A consistent set of middleware components such as RTOS, FAT file system, and graphics
  - All embedded software utilities with full sets of peripheral and applicative examples

Figure 1. STM32Cube block diagram



a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Global architecture

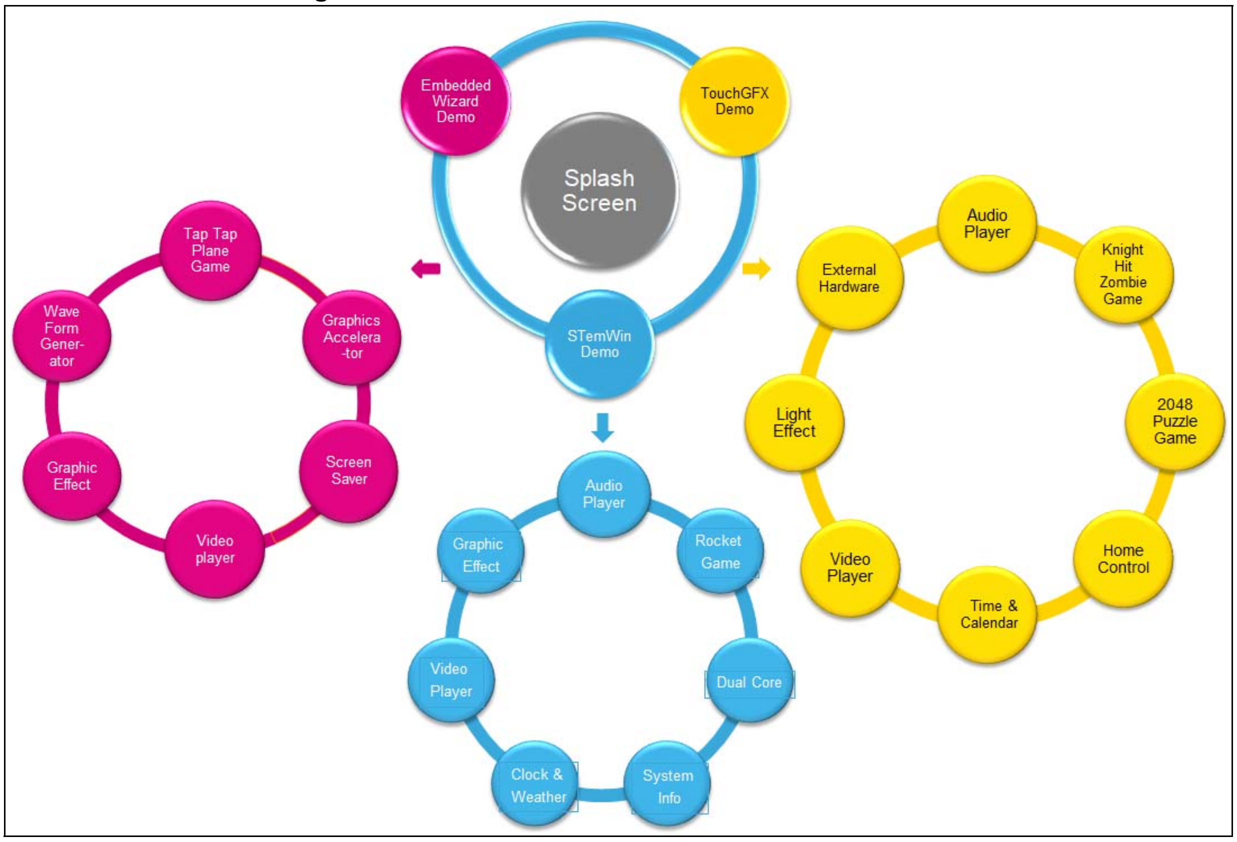
The STM32CubeH7 demonstration is composed of a central kernel based on a set of firmware and hardware services offered by the STM32Cube middleware, several Evaluation and Discovery boards, and a set of modules mounted on the kernel and built in a modular architecture. Each module is separately reusable in a standalone application.

The kernel manages the full set of modules. It provides access to all common resources and facilitates the addition of new modules as shown in *Figure 2*.

Each module provides the following functionalities and properties:

- Icon and graphical aspect characteristics
- Method to start up the module
- Method to close down safely the module (such as Hot unplug for unit storage)
- Method to manage low-power modes
- The module application core (main module process)
- Specific configuration
- Error management

Figure 2. STM32CubeH7 demonstration architecture



3 Kernel description

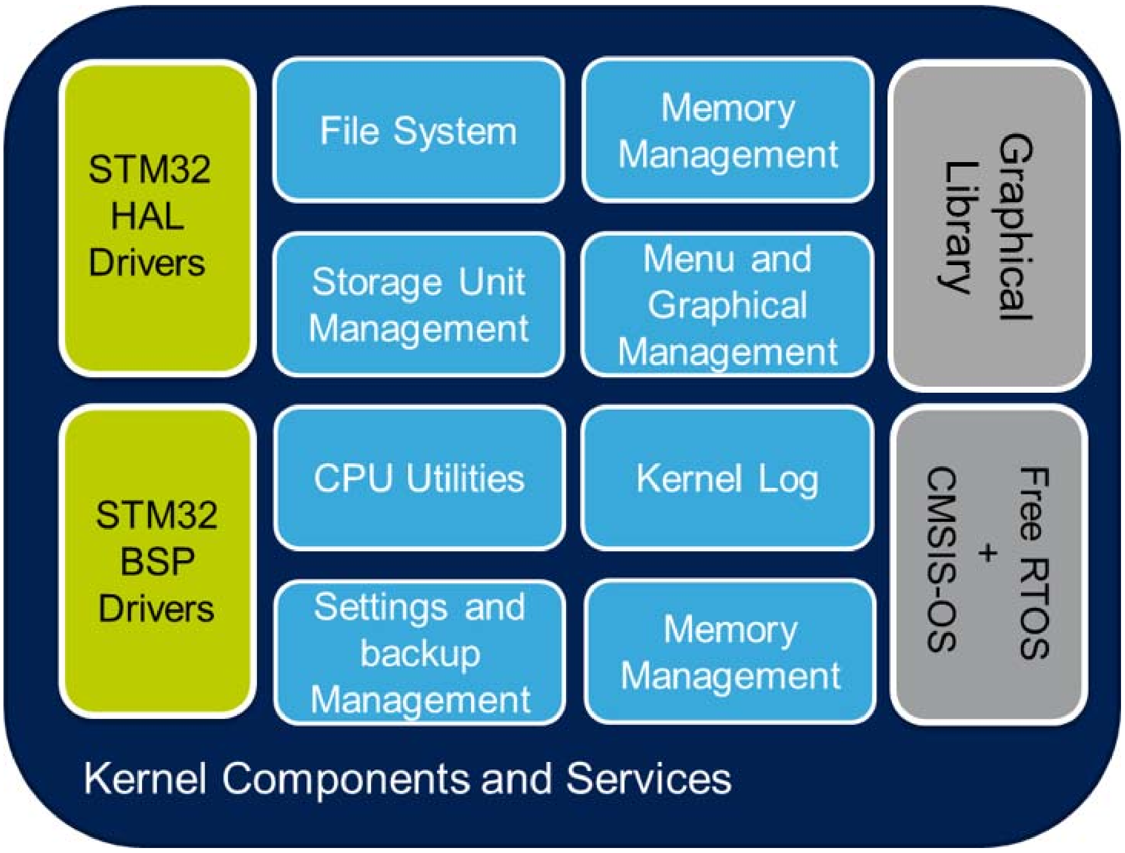
3.1 Overview

The role of the demonstration kernel is mainly to provide a generic platform that controls and monitors all the application processes. The kernel provides a set of friendly user APIs and services, allowing the user modules to benefit from all the hardware and firmware resources.

The kernel provides the tasks and services listed below:

- Hardware and modules initialization:
  - BSP initialization (SDRAM, touch screen, CRC, RTC, Quad-SPI)
  - GUI initialization
- Memory management
- Graphical resources and main menu management
- Storage management
- System monitoring and settings
- CPU utilities (CPU usage, running tasks)

Figure 3. Kernel components and services



3.2 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 by the middleware components:

- Touch screen
- SDRAM
- Quad-SPI Flash memory
- Backup SRAM
- 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 units
- Modules manager
- Kernel log

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

3.3 Kernel processes and tasks

The kernel is composed of a main task and software timer scheduled by FreeRTOS through the CMSIS-OS wrapping layer:

- **GUI thread**: this task initializes the demonstration main menu and then handles the graphical background task when requested by the STemWin.

```c
/**
  * @brief  Start task
  * @param  argument: pointer that is passed to the thread function as start argument.
  * @retval None
  */
static void GUIThread(void const * argument)
{
  /* Initialize Storage Units */
  k_StorageInit();

  /* Initialize GUI */
  GUI_Init();

  GUI_X_InitOS();

  /* Enable Multibuffereing and set Layer0 as the default display layer */
  WM_MULTIBUF_Enable(1);
  GUI_SetLayerVisEx (1, 0);
  GUI_SelectLayer(0);

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

  /* Display immediatly the Menu */
  GUI_Exec();

  /* Gui background Task */
  while(1) {
    osSemaphoreWait( LcdUpdateSemaphoreId , 1000 );
    GUI_Exec();
  }
}
```

- **Software timer**: managing periodically (each 30 ms) the touch screen state.

```c
/**
  * @brief  Timer callbacsk (30 ms)
  * @param  n: Timer index
  * @retval None
  */
static void TimerCallback(void const *n)
{
  k_TouchUpdate();
}

```

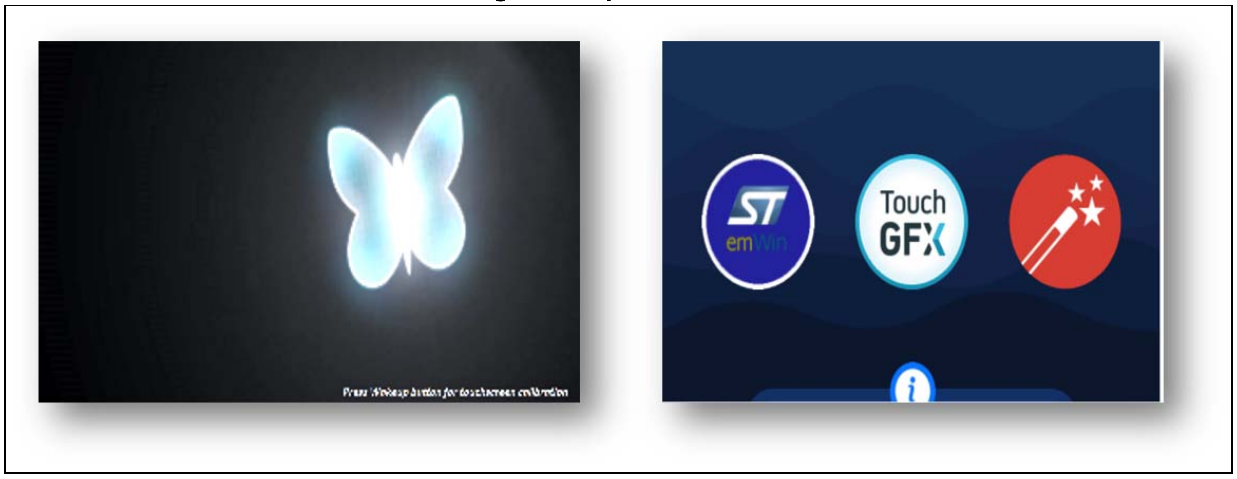
Note: ***For the STM32H747I-EVAL and STM32H747I-DISCO board demonstrations, the interrupt handles the touch screen event.***

3.4 Kernel graphical aspect

The graphical aspect of the STM32Cube demonstration is divided into three main graphical components listed below:

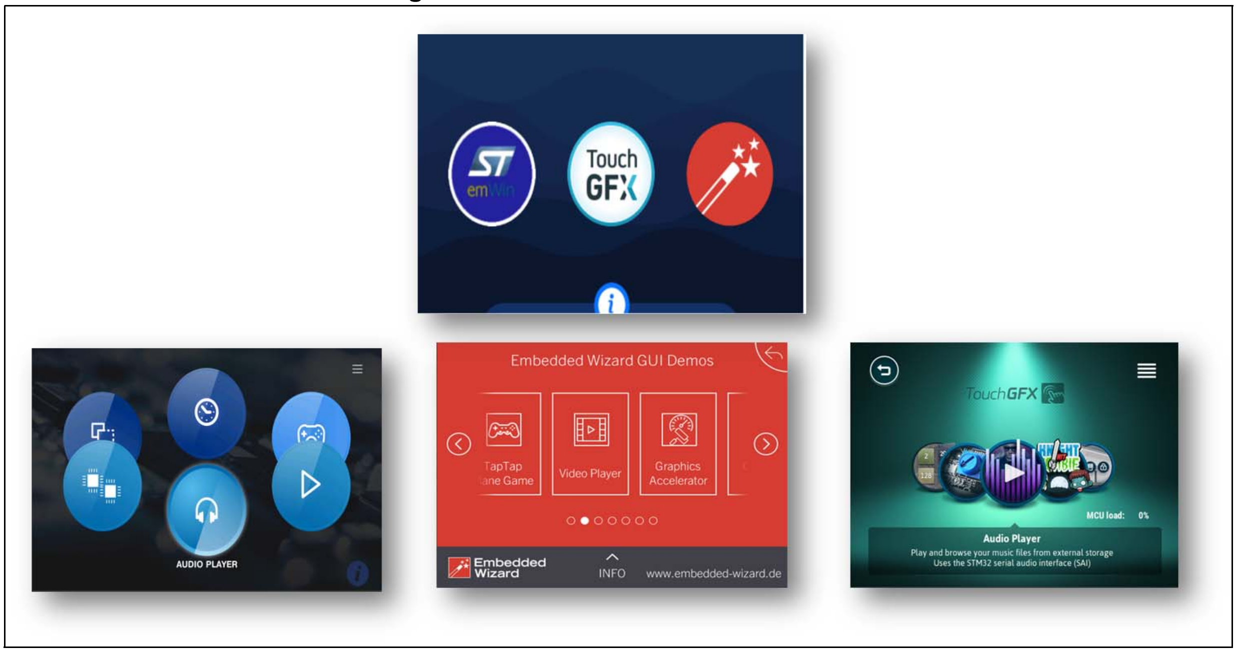
- At the board reset, the **splash screen** appears for a few seconds. The splash screen can be skipped by a simple click on the screen, to launch the main menu of the three demonstrations. In the case of the STM32H747I-DISC1 (Discovery board provided without LCD screen board), the demonstration software ends with blanking the red LED LD3.

Figure 4. Splash screen



- Start the main menu of each demonstration by clicking on the dedicated icon.

Figure 5. Demonstrations main menu



Note: STemWin, TouchGFX, and the source code of the menu launcher are available. The embedded wizard demonstration is available only with the full binary file.

3.5 Kernel menu management

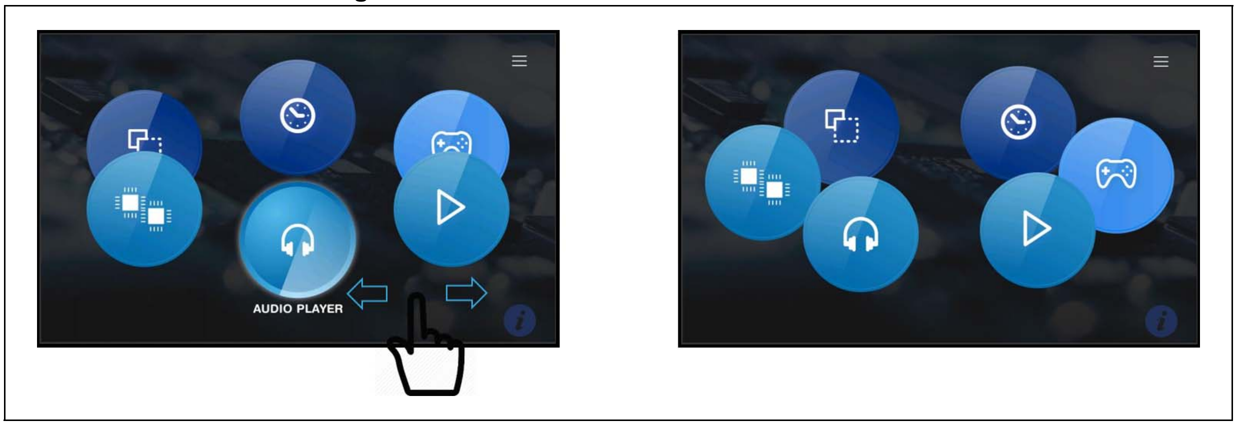
Note: ***Important*** *- This user manual describes all the demonstration modules. STemWin and TouchGFX modules are detailed, as the source code is provided.*

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

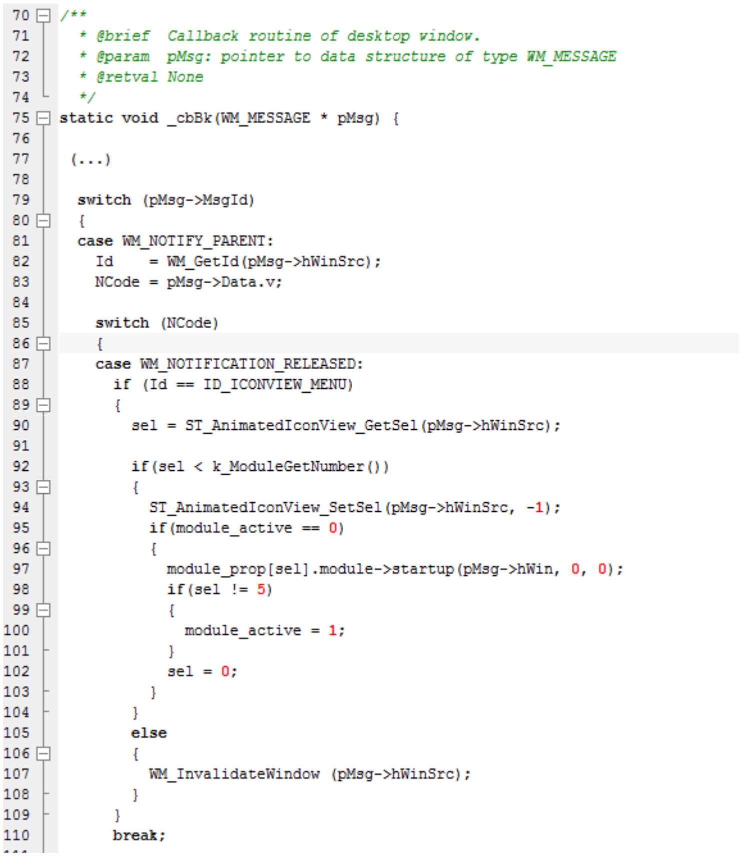
- Draw the background image.
- Restore general settings from backup memory.
- Setup the main desktop callback to manage main window messages.

The icon view widget contains the icons associated with added modules. The user can launch a module by a simple click. The user can also slide the icons and select the modules.

Figure 6. STemWin demonstration main menu



A module is launched with a simple click on the associated icon by calling the startup function in the module structure. This is performed when a `WM_NOTIFICATION_RELEASED` message arrives at the desktop callback with `ID_ICONVIEW_MENU`.

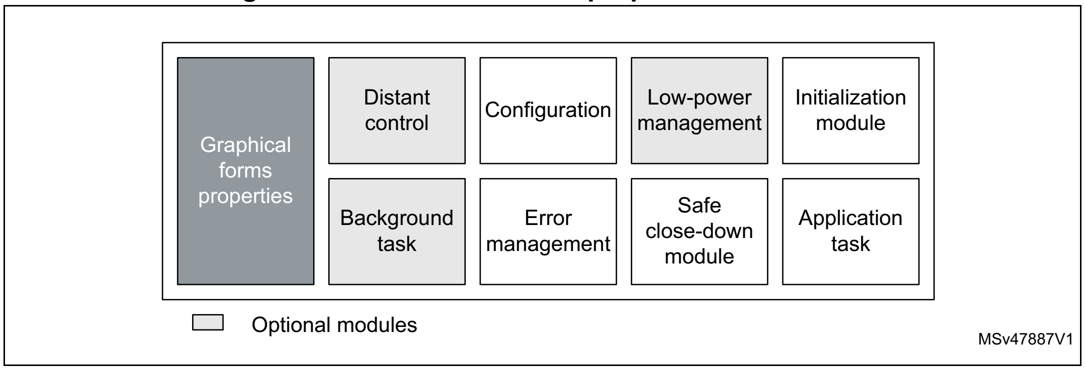


```
/**
 * @brief  Callback routine of desktop window.
 * @param  pMsg: pointer to data structure of type WM_MESSAGE
 * @retval None
 */
static void _cbBk(WM_MESSAGE * pMsg) {

 (...)

  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)
      {
        sel = ST_AnimatedIconView_GetSel(pMsg->hWinSrc);

        if(sel < k_ModuleGetNumber())
        {
          ST_AnimatedIconView_SetSel(pMsg->hWinSrc, -1);
          if(module_active == 0)
          {
            module_prop[sel].module->startup(pMsg->hWin, 0, 0);
            if(sel != 5)
            {
              module_active = 1;
            }
            sel = 0;
          }
        }
        else
        {
          WM_InvalidateWindow (pMsg->hWinSrc);
        }
      }
      break;
```

3.6 Module manager

The kernel manages the modules. It is responsible for initializing relative hardware and GUI resources, common resources such as a storage unit, graphical widget and the system menu.

Figure 7. Functionalities and properties of modules



Each module provides the following functionalities and properties:

- Icon and graphical component structure
- Method to startup the module
- Method to close down safely the module (such as Hot unplug for MS Flash disk)
- Method to manage low-power modes (optional)
- Application task
- Module background process (optional)
- Remote control method (optional)
- Specific configuration
- Error management

The modules can be added to the demonstration and use the common kernel resources. The code below shows how to add a module to the demonstration.

```
/* Initialise Modules function */
k_ModuleInit();

/* Link modules */
k_ModuleAdd(&audio_player_board);
k_ModuleAdd(&video_player_board);
k_ModuleAdd(&rocket_game_board);
k_ModuleAdd(&analog_clock_board);
k_ModuleAdd(&graphic_effects_board);
k_ModuleAdd(&dual_core_board);
```

A module is a set of functions and data structures, which are defined in a global data structure, and provide all the information and pointers to specific methods and functions to the kernel. This later checks the integrity and the validity of the module and inserts its structure into a module table. A unique identifier (UID) identifies each module. When two modules have the same UID, the kernel rejects the second one.

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

The module structure is defined as follows:

- id: unique module identifier
- name: pointer to the module name
- open_icon: pointer to module icon frames (array of bitmap format moving on the right)
- close_icon: pointer to module icon (array of bitmap format moving on the left)
- startup: the function creating the module frame and control buttons
- DirectOpen: reserved for feature use.

3.7 Backup and settings configuration

The STM32Cube demonstration saves the kernel and module settings using the following method:

- Using the RTC backup register (32-bit data width): the data to save must be a 32-bit data and can be defined as a bit field structure.

```
typedef union
{
  uint32_t d32;
  struct
  {
    uint32_t repeat         : 2;
    uint32_t pause          : 1;
    uint32_t play           : 1;
    uint32_t stop           : 1;
    uint32_t mute           : 1;
    uint32_t volume         : 8;
    uint32_t reserved       : 18;
  }b;
}
AudioSettingsTypeDef;
```

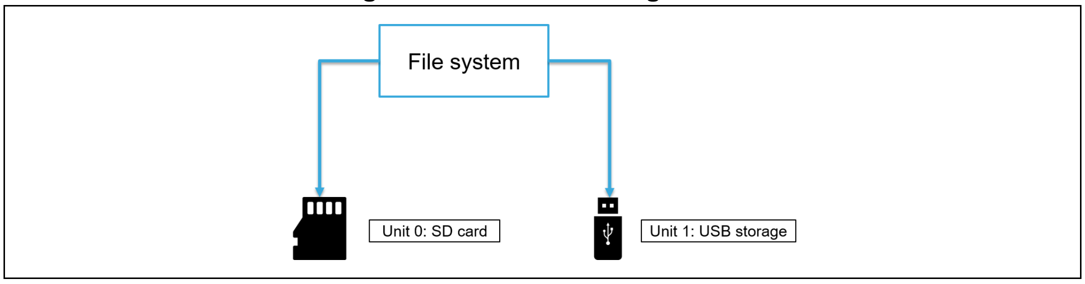
- Two kernel APIs are used to save or restore the structure from the RTC backup registers.

```
void     k_BkupSaveParameter(uint32_t address, uint32_t data);
uint32_t k_BkupRestoreParameter(uint32_t address);
```

3.8 Storage units

The STM32Cube demonstration kernel offers a storage unit to retrieve audio, bitmaps and video media. The storage unit is initialized during the platform startup and thus it is available to all the modules during the STM32Cube Demonstration run time.

Figure 8. Available storage units



The unit is accessible through the standard I/O operations offered by the FatFS used in the development platform. The unit is mounted automatically when the physical media is connected to the connector on the board. *Table 1* lists the file system interface functions (FatFS functions) used to deal with the physical storage unit.

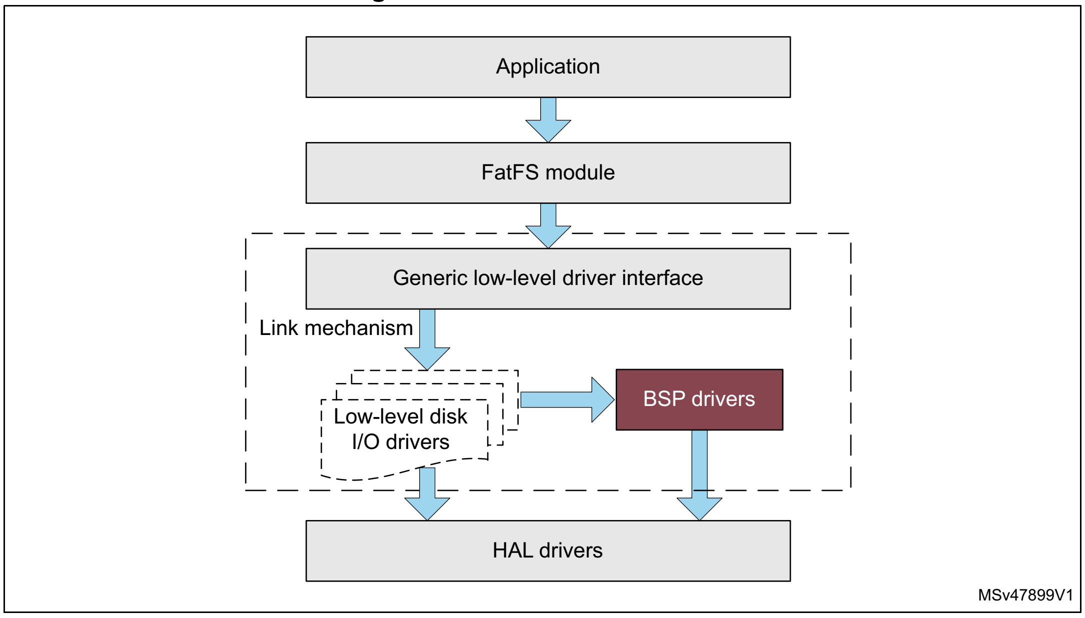
Table 1. File system interface functions

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 the unit is ready
disk_ioctl	Control device-dependent features
f_mount	Register / unregister a work area
f_open	Open / create a file
f_close	Close a file
f_read	Read a file
f_write	Write a 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 a file status
f_mkdir	Create a directory
f_unlink	Remove a file or a directory
f_chmod	Change attribute
f_utime	Change timestamp
f_rename	Rename/move a file or a 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

In the FatFS file system, the page size is fixed to 512 bytes. The SD cards with higher page sizes are not supported.

The software architecture of the storage unit is described in *Figure 9*.

Figure 9. Software architecture



The FatFS is mounted upon the mass storage to allow abstract access to the physical media through standard I/O methods.

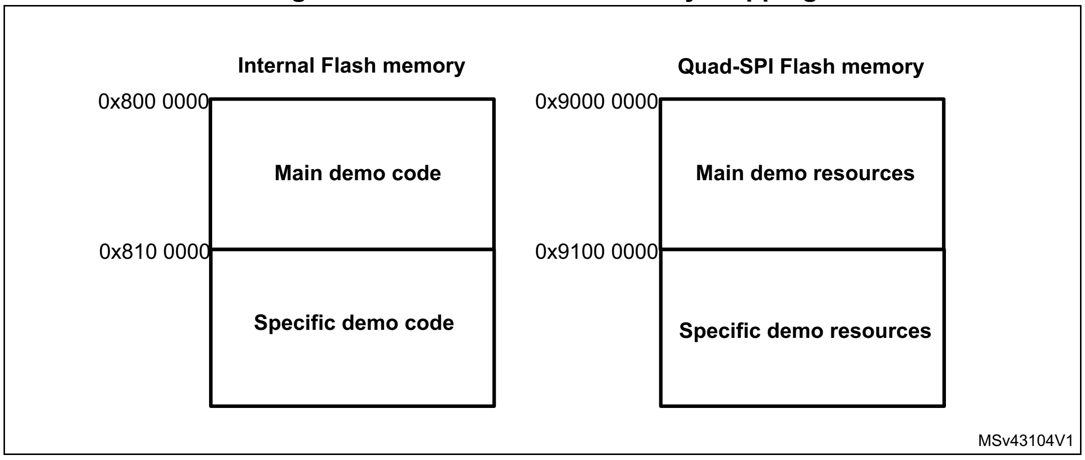
3.9 Adding binary demonstration

The user can load a specific demonstration as a binary in a specific memory address. The specific demonstration is launched during the run-time of the native ST demonstration. The main demonstration (ST demonstration) jumps to the specific demonstration address. From the specific demonstration, the user can go back to the main demonstration by doing a hardware reset.

The specific demonstration must provide a control button named “Menu” that triggers a hardware reset and saves a specific signature in the backup SRAM.

Figure 10 shows how the main demonstration and the specific demonstration must be mapped in the memory.

Figure 10. Demonstration memory mapping



Main demonstration

Upon clicking on the specific demonstration icon in the main menu of the native main demonstration, a signature A is saved in the backup SRAM and a reset is performed.

During the next start of the ST demonstration, the signature is checked. If the result is A, the PC jumps to the specific demonstration memory location and the specific demonstration starts.

Specific demonstration

The specific demonstration must provide a GUI control button named ‘Menu’. When ‘Menu’ is activated, a signature B is saved in the backup SRAM and a reset is performed.

During the next start, the startup screen is bypassed and the main demonstration menu is directly shown.

Signature and base address

```
#define SPECIFC_DEMO_ADDRESS 0x08100000
#define SPECIFC_DEMO_SIGNATURE_A 0x5AA55AAA
#define SPECIFC_DEMO_SIGNATURE_B 0x5AA55BBB
```

Reset sequence

The reset sequence must be built as follows:

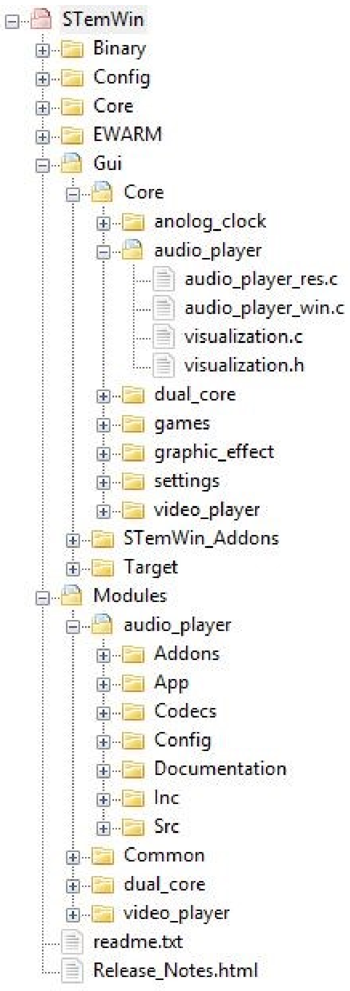
```
__HAL_RCC_RTC_ENABLE();
__HAL_RCC_PWR_CLK_ENABLE();
__HAL_RCC_BKPSRAM_CLK_ENABLE();
HAL_PWR_EnableBkUpAccess();
(...)
*(uint32_t *)(0x40024000) = SPECIFIC_DEMO_SIGNATURE_B;
NVIC_SystemReset();
```

In the system_stm32h7xx.c, the specific demonstration must change the vector table offset define (`#define VECT_TAB_OFFSET`) to 0x100000. The *system_init* function then sets the VTOR (vector table offset register) to the specific demonstration base address (0x08100000).

3.10 Demonstration repository

The STM32Cube is a component in the STM32Cube package. *Figure 11* shows the demonstration folder organization.

Figure 11. Demonstration folder structure



```
STemWin
    Binary
    Config
    Core
    EWARM
    Gui
        Core
            anolog_clock
            audio_player
                audio_player_res.c
                audio_player_win.c
                visualization.c
                visualization.h
            dual_core
            games
            graphic_effect
            settings
            video_player
        STemWin_Addons
        Target
    Modules
        audio_player
            Addons
            App
            Codecs
            Config
            Documentation
            Inc
            Src
        Common
        dual_core
        video_player
    readme.txt
    Release_Notes.html
```

In the STM32Cube package, the demonstration sources are located in the *Demonstration* folder of each supported board. The sources are divided into groups described as follows:

- Core: contains the kernel files
- GUI: contains the module core manager, the graphical aspect and the windowing management of the modules. It contains also the binary file for added widgets based on STemWin graphical library.
- Config: contains all components of the middleware and HAL configuration files
- Modules: contains the applicative part of the modules
- Binary: demonstration binary file in hex format
- Project settings: a folder per toolchain containing the project settings and the linker files.

3.11 Kernel components

Table 2. Kernel components list

Function	Description
Kernel core	Kernel core and utilities
Modules	User and system modules
STM32 HAL Drivers	STM32Cube HAL driver relative to the STM32 device used
BSP drivers	Evaluation board (or Discovery kit) BSP drivers
CMSIS	CMSIS Cortex-M7 device peripheral access layer system source file
FatFS	FatFS file system
FreeRTOS	FreeRTOS real-time operating system
STemWin	STemWin graphical library

3.12 Kernel core files

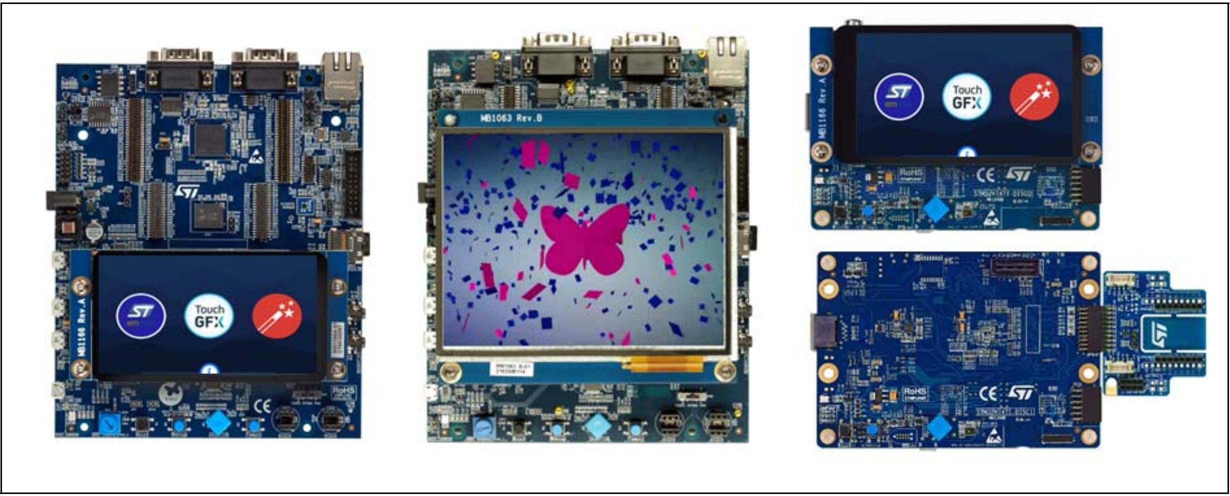
Table 3. Kernel core files list

Function	Description
main.c	Main program file
stm32h7xx_it.c	Interrupt handlers for the application
k_bsp.c	Provides the kernel BSP functions
k_menu.c	Kernel menu and desktop manager
k_module.c	Modules manager
stm32h7xx_hal_timebase_tim.c	Use the hardware timer to configure the time base
k_rtc.c	RTC and backup manager
k_startup.c	Demonstration startup windowing process
k_storage	Storage units manager
startup_stm32h7xxxx.s	Startup file
cpu_utils.c	CPU load calculation utility

3.13 Hardware settings

The STM32Cube demonstration supports STM32H7 Series devices and runs on STM32H743I-EVAL, STM32H747I-EVAL, STM32H747I-DISCO, and STM32H747I-DISC1 boards from STMicroelectronics.

Figure 12. STM32Cube demonstration boards



1. Pictures are not contractual.

Table 4. Jumpers for demonstration boards

Board	Jumper	Position description
STM32H743I-EVAL and STM32H747I-EVAL	JP3	Status: not fitted The Bootloader_BOOT is managed by pin 6 of connector CN7 (RS232 DSR signal) when JP1 is closed. This configuration is used for boot loader application only.
	JP9	Status: fitted Used to measure MCU current consumption manually
	JP10	Position: PSU position. For power supply jack (CN10)
STM32H747I-DISCO and STM32H747I-DISC1	JP3	Status: fitted Used to measure MCU current consumption manually
	JP6	Position: STlk (STLINK) Used to select the power supply source.

4 How to create a new module

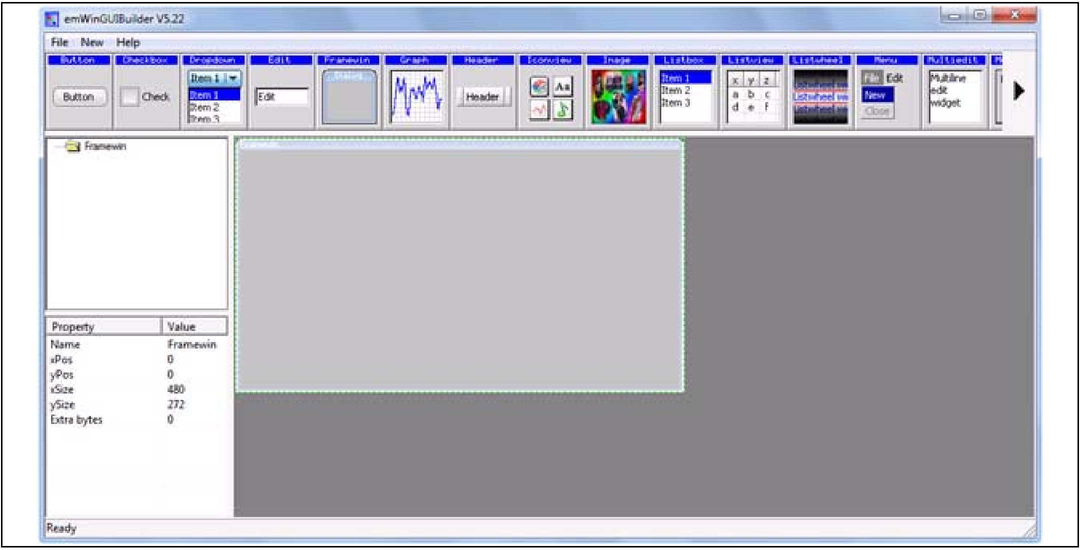
A module is composed of two main parts: the graphical aspect and the set of associated functionalities.

4.1 Creating the graphical aspect

The graphical aspect consists of the mainframe window plus the full set of the visual elements (such as buttons, checkboxes and progress bars), used to control and monitor functionalities of the modules.

A very useful PC application, the GUIBuilder, provided into the demonstration package, allows easy and quick creation of the module frame window and all its components.

Figure 13. GUIBuilder overview


emWinGUIBuilder V5.22
File New Help
Button Checkbox Dropdown Edit Framewin Graph Header Iconview Image Listbox Listview Listwheel Menu Multiedit
Button Check Item 1 Item 2 Item 3 Edit Header
Framewin
Ready

Property	Value
Name	Framewin
xPos	0
yPos	0
xSize	480
ySize	272
Extra bytes	0

It only takes a few minutes to completely design the module appearance using “drag and drop” commands and then to generate a source code file to be included totally or partially into the application.

The file generated is composed of the two main parts listed below:

- A resource table: **GUI_WIDGET_CREATE_INFO** type of table, which specifies all the widgets to be included in the dialog and also their respective positions and sizes
- A dialog procedure: described more in detail in section **Error! Reference source not found** (it is referred to as “main module callback routine”).

4.2 Graphics customization

After the creation of the basic module graphical appearance, it is 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 must be created and used instead of the original one.

Here is an example of a custom callback for the Play button.

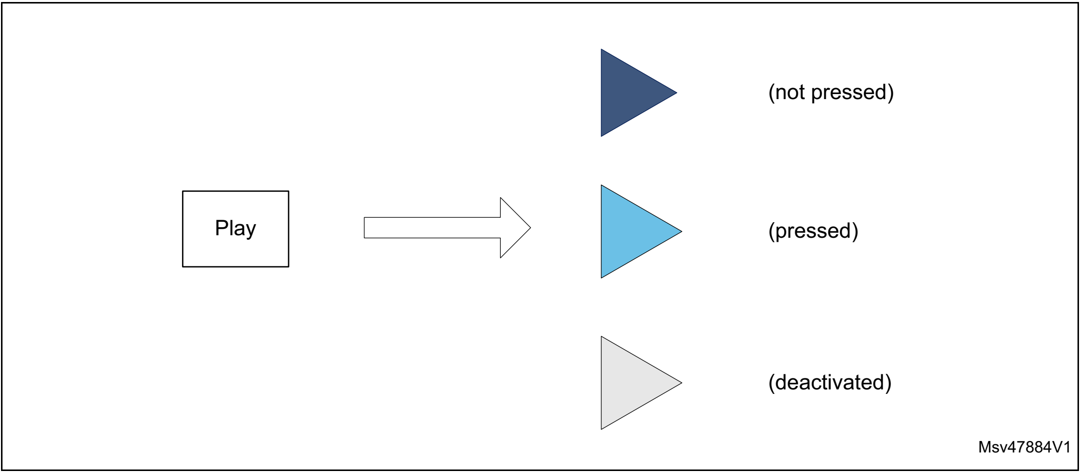
```
/**
  * @brief  callback for play button
  * @param  pMsg: pointer to data structure of type WM_MESSAGE
  * @retval None
  */
static void _cbButton_play(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
    case WM_PAINT:
      _OnPaint_play(pMsg->hWin, PlayerSettings.b.pause);
      break;
    default: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

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

Obviously, the new callback must be associated with the graphical element (in our case the Play button) when it is created, like below.

```
    hItem = BUTTON_CreateEx(112, 420, 40, 40, pMsg->hWin, WM_CF_SHOW, 0, ID_PLAY_BUTTON);
    WM_SetCallback(hItem, _cbButton_play);
```

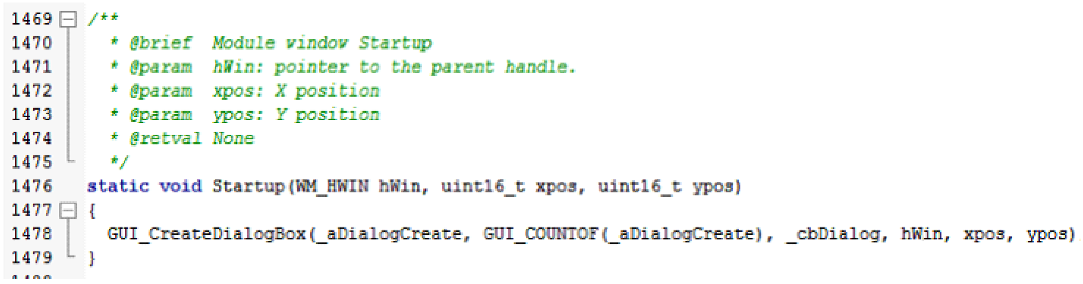
Figure 14. Graphics customization



4.3 Module implementation

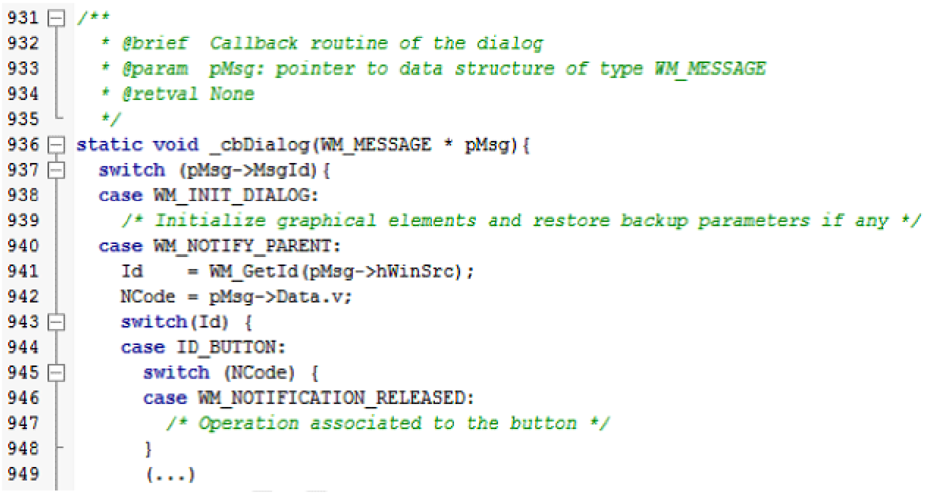
Once the graphical part of the module is finalized, the module functionalities and processes can be added.

It begins with the creation of the main module structure as defined in *Section 3.6: Module manager*. Each module has its own Startup function that 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 as follows.



```
/**
  * @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 window messages presented in the code portion above (WM_INIT_DIALOG and WM_NOTIFY_PARENT) is not exhaustive, but represents the essential message IDs used:

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

The full list of window messages is available in the ***WM.h*** file.

4.4 Adding a module to the main desktop

Once the appearance and functionality of the module are defined and created, the module 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, the `k_ModuleAdd()` function must 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*.

5 Demonstration customization and configuration

5.1 LCD configuration

The LCD is configured through the *LCDConf.c* file. Amongst the several parameters that can be set in this file, some are detailed below:

- Multiple layers:

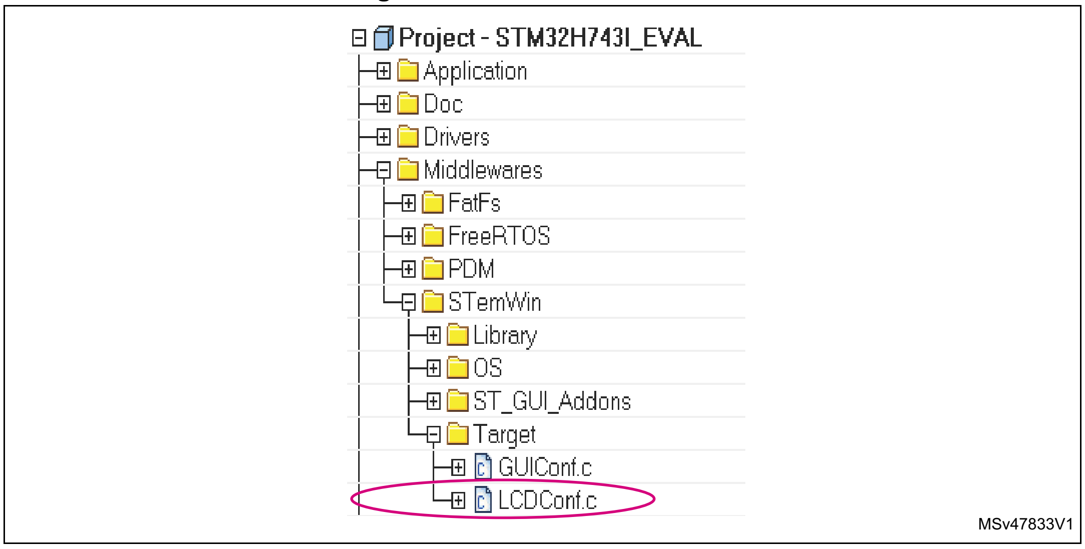
  The number of layers used is defined using GUI_NUM_LAYERS. Its value must not exceed the one defined into *GUIConf.h* (the later represents the maximum number of available layers supported when the STemWin binary is generated).
- Multiple buffering:

  If NUM_BUFFERS is set to a value ‘n’ greater than 1, it means that ‘n’ frame buffers are used for drawing operation (see section **Error! Reference source not found** for the impact of multiple buffering on performance).
- Virtual screens:

  If the display area is greater than the physical size of the LCD, NUM_VSCREENS must be set to a value greater than 1. Note that virtual screens and multi buffers are not allowed together.
- Frame buffers locations:

  The physical location of the frame buffer is defined through LCD_LAYERX_FRAME_BUFFER.

Figure 15. LCDConf.c location



```
Project - STM32H743I_EVAL
├─ Application
├─ Doc
├─ Drivers
├─ Middlewares
│  ├─ FatFs
│  ├─ FreeRTOS
│  ├─ PDM
│  └─ STemWin
│     ├─ Library
│     ├─ OS
│     ├─ ST_GUI_Addons
│     └─ Target
│        ├─ GUIConf.c
│        └─ LCDConf.c
```

5.2 Layer management

In the demonstration package, GUI_NUM_LAYERS is set to 2 (both layers are used):

- Layer 0 is used for the main desktop display.
- Layer 1 is used for video player module playback.

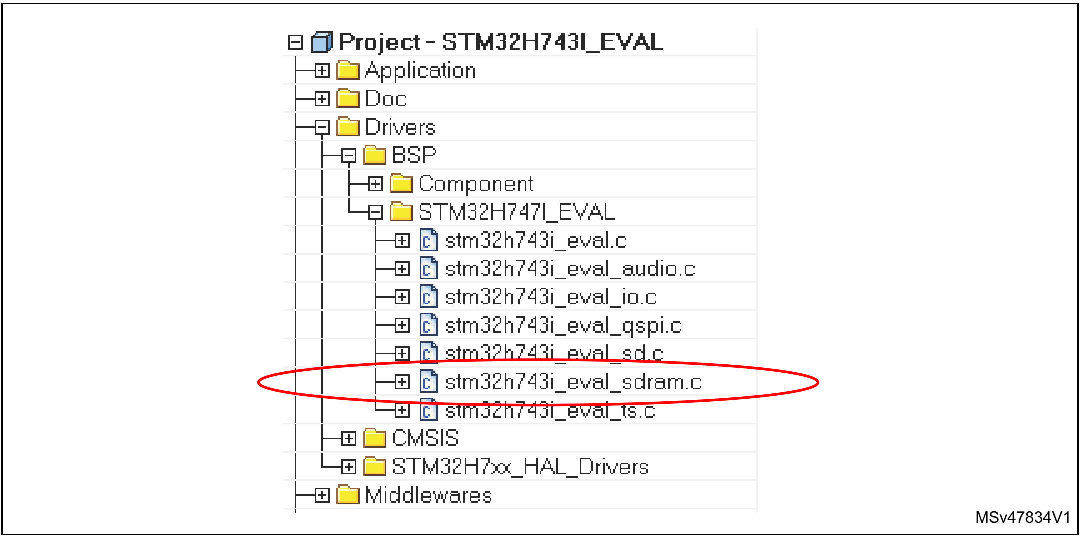
Such display separation helps to lighten the CPU usage during the refresh tasks.

5.3 BSP customization

5.3.1 SDRAM configuration

The BSP SDRAM driver offers a set of functions to initialize, read/write in polling mode or DMA mode.

Figure 16. SDRAM initialization



```
⊟ Project - STM32H743I_EVAL
├⊞ Application
├⊞ Doc
├⊟ Drivers
│ ├⊟ BSP
│ │ ├⊞ Component
│ │ └⊟ STM32H747I_EVAL
│ │   ├⊞ stm32h743i_eval.c
│ │   ├⊞ stm32h743i_eval_audio.c
│ │   ├⊞ stm32h743i_eval_io.c
│ │   ├⊞ stm32h743i_eval_qspi.c
│ │   ├⊞ stm32h743i_eval_sd.c
│ │   ├⊞ stm32h743i_eval_sdram.c
│ │   └⊞ stm32h743i_eval_ts.c
│ ├⊞ CMSIS
│ └⊞ STM32H7xx_HAL_Drivers
├⊞ Middlewares
```

The SDRAM external memory must be initialized before the GUI initialization to allow the SDRAM to be used as an LCD layers frame buffer.

Table 5. LCD frame buffer locations

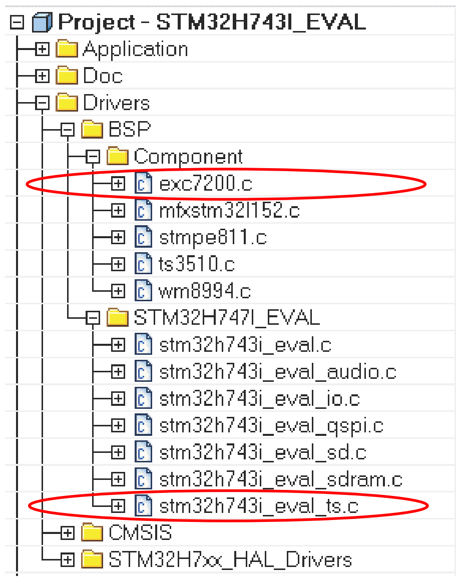
Layer	Address
LCD layer 0	0xD000 0000
LCD layer 1	0xD0000000 + (size of frame buffer * NUM_VSCREENS[1] * NUM_BUFFERS)[2]

1. NUM_VSCREENS: number of virtual screens defined in *LCDConf.c* file.
2. NUM_BUFFERS: number of multi buffers defined in *LCDConf.c* file.

5.3.2 Touchscreen configuration

The touchscreen is controlled by the BSP TS driver, which uses the exc7200 component in the case of the STM32H743I-EVAL board, and the ft6x06 component for the STM32H747I-EVAL and STM32H747I-DISCO boards.

Figure 17. Touch screen configuration



```
Project - STM32H743I_EVAL
├─ Application
├─ Doc
├─ Drivers
│  ├─ BSP
│  │  ├─ Component
│  │  │  ├─ exc7200.c
│  │  │  ├─ mfxstm32l152.c
│  │  │  ├─ stmpe811.c
│  │  │  ├─ ts3510.c
│  │  │  └─ wm8994.c
│  │  └─ STM32H747I_EVAL
│  │     ├─ stm32h743i_eval.c
│  │     ├─ stm32h743i_eval_audio.c
│  │     ├─ stm32h743i_eval_io.c
│  │     ├─ stm32h743i_eval_qspi.c
│  │     ├─ stm32h743i_eval_sd.c
│  │     ├─ stm32h743i_eval_sdram.c
│  │     └─ stm32h743i_eval_ts.c
│  ├─ CMSIS
│  └─ STM32H7xx_HAL_Drivers
```

MSv47835V1

The touch screen is initialized in `k_BspInit` following the used screen resolution as shown in the code below.

```
/**
  * @brief  Initializes LEDs, SDRAM, touch screen, CRC and SRAM.
  * @param  None
  * @retval None
  */
void k_BspInit(void)
{
  ( ... )

  /* Initialize the Touch screen */
  BSP_TS_Init(640, 480);

  ( ... )
}

/**
  * @brief  Read the coordinate of the point touched and assign their
  *         value to the variables u32_TSXCoordinate and u32_TSYCoordinate
  * @param  None
  * @retval None
  */
void k_TouchUpdate(void)
{
  static GUI_PID_STATE TS_State = {0, 0, 0, 0};

  BSP_TS_GetState((TS_StateTypeDef *)&ts);

  ( ... )

  GUI_TOUCH_StoreStateEx(&TS_State);

  ( ... )

}
```

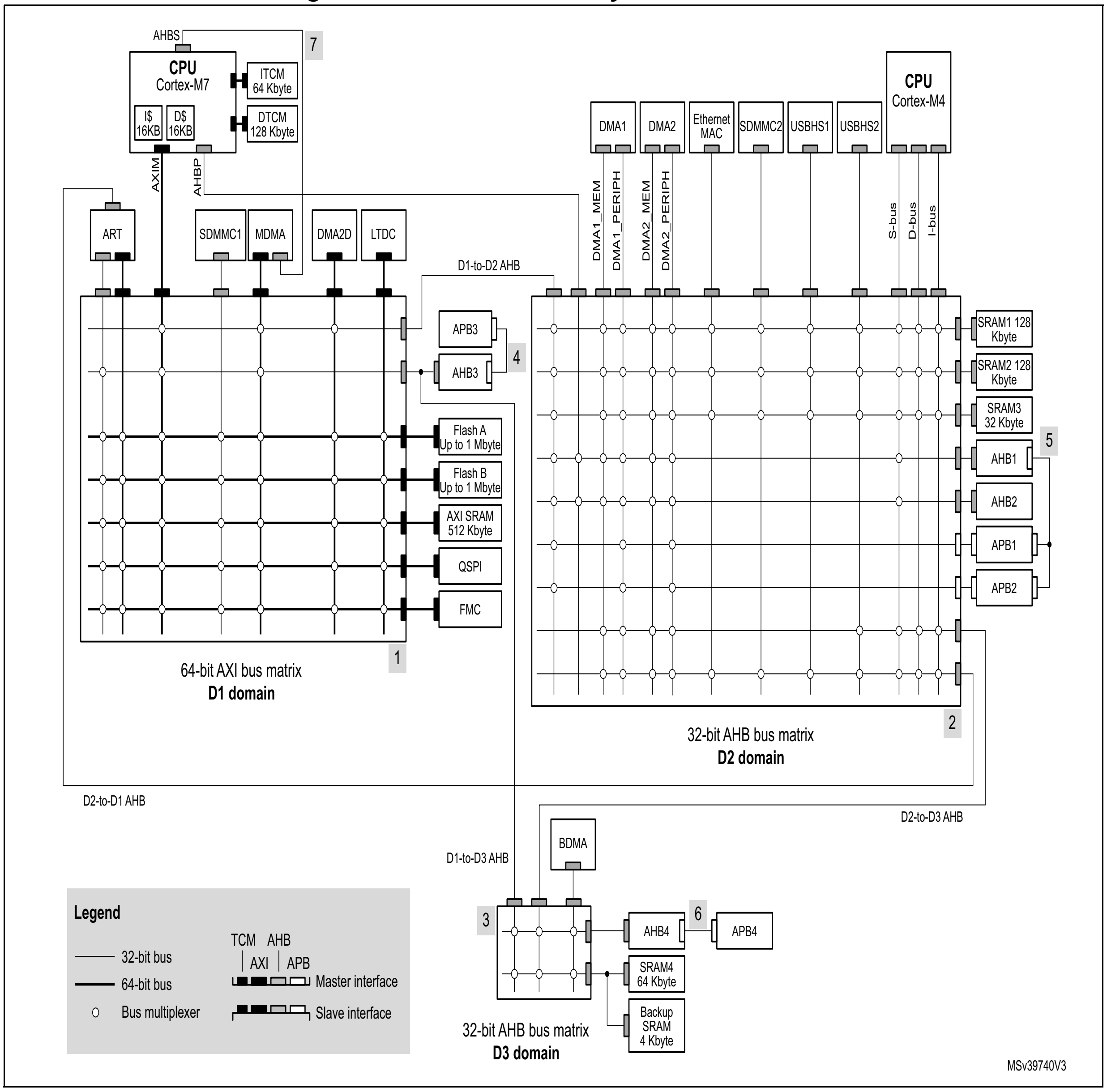
6 Performance

6.1 CPU cache

The STM32CubeH7 demonstration takes benefit from Cortex-M7 performance:

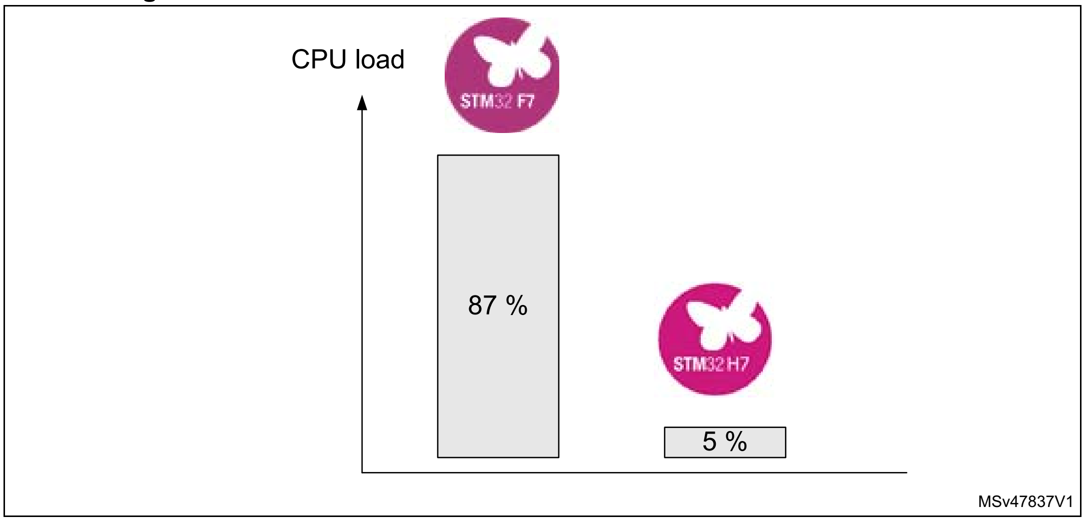
- 16 Kbytes dedicated to instruction cache
- 16 Kbytes dedicated to data cache

Figure 18. STM32H7 Series system architecture

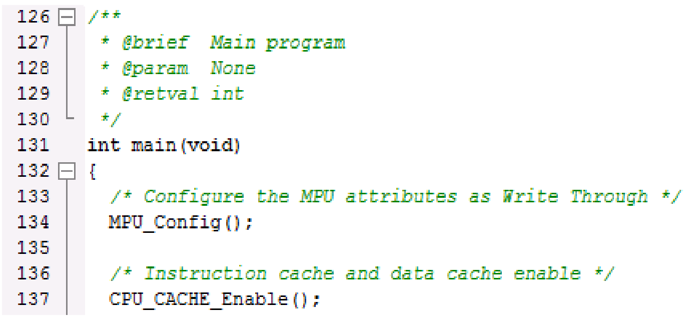


Using the STM32H7 Series, the CPU load is decreased with video module from 87% to 5% compared to STM32F7 Series, thanks to the hardware JPEG decoding and the support of hardware conversion from YCbCr to RGB using Chrom-ART Accelerator.

Figure 19. Performance of STM32H7 Series versus STM32F7 Series



The instruction cache and data cache are enabled in the *main.c* file as shown in the code below.

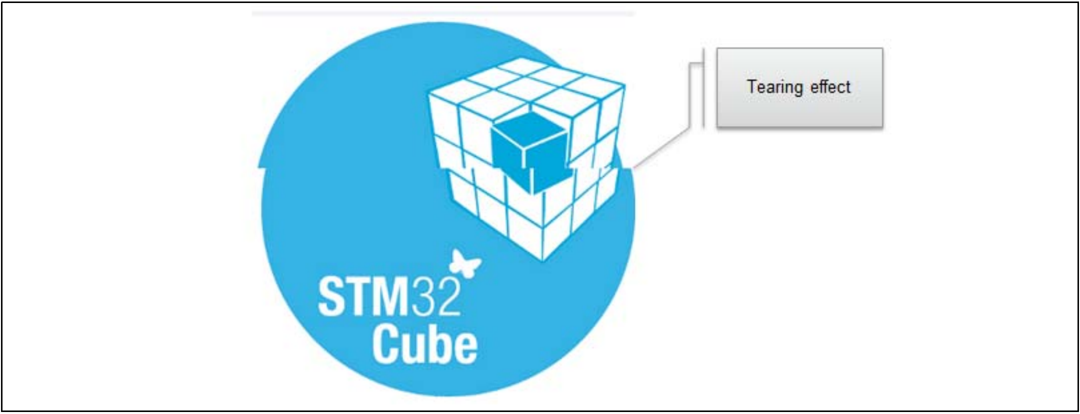


```
/**
  * @brief  Main program
  * @param  None
  * @retval int
  */
int main(void)
{
  /* Configure the MPU attributes as Write Through */
  MPU_Config();

  /* Instruction cache and data cache enable */
  CPU_CACHE_Enable();
```

6.2 Multi buffering features

The multiple buffering is the use of more than one frame buffer so that the display ever shows a screen that is already completely rendered, even if a drawing operation is in process. When starting the process of drawing, the current content of the front buffer is copied into a back buffer. After that, all drawing operations take effect only on this back buffer. After the drawing operation has been completed, the back buffer becomes the front buffer. Making the back buffer the visible front buffer normally only requires the modification of the start address in the frame buffer register of the display controller. Now it must be considered that the display refreshes a display approximately 60 times per second. After each period, there is a vertical synchronization signal, known as the VSYNC signal. The best moment to make the back buffer the new front buffer is this signal. If not considering the VSYNC signal, tearing effects may occur, as shown in *Figure 20* below.

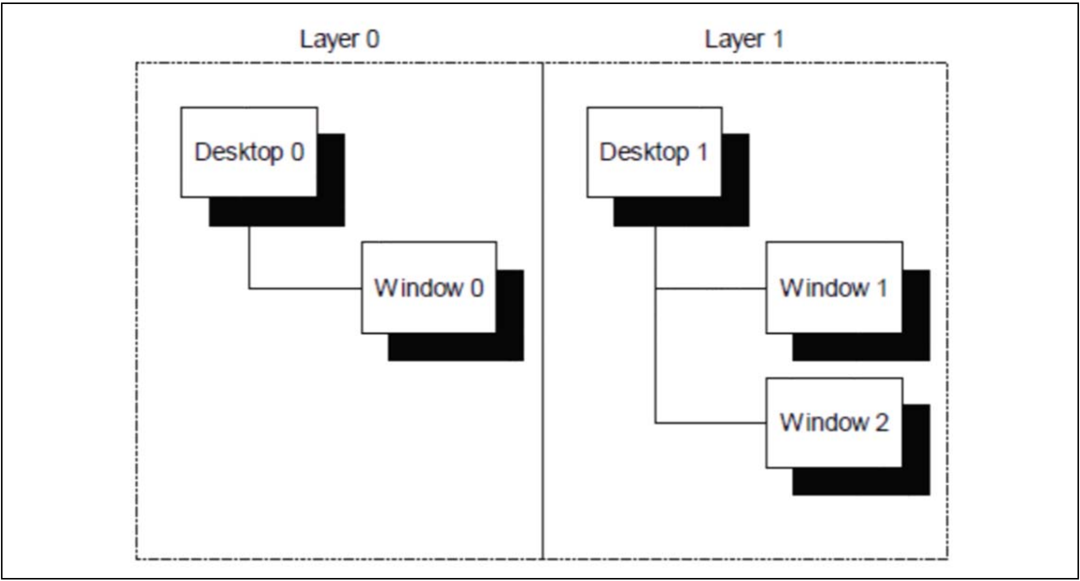
Figure 20. Example of tearing effect



6.3 Multi-layers feature

The windows can be placed in any layer or display. Drawing operations is usable on any layer or display. Since there are only small differences from this point of view, multiple layers and multiple displays are handled in the same way (using the same API routines) and are simply referred to as multiple layers, even if the particular embedded system uses multiple displays.

Figure 21. Independent layer management



6.4 Hardware acceleration

With the STM32H7 demonstrations, the hardware acceleration capabilities of the STM32H7 devices are used. STemWin offers a set of customization callbacks to change the default behavior based on the hardware capabilities.

The optimized processes are implemented in the *LCDConf.c* file, with the following features:

- Color conversion

  STemWin works internally with logical colors (ABGR). To be able to translate these values into index values for the hardware and vice versa, the color conversion routines automatically use the DMA2D for that operation if the layer works with direct color mode. This low-level implementation secures that the DMA2D is used each time multiple colors or index values need to be converted.
- Drawing of index-based bitmaps

  When drawing index-based bitmaps, STemWin first loads the bitmap palette into the DMA2Ds LUT (lookup table) instead of directly translating the palette into index values for the hardware. Then, one single DMA2D function call performs the drawing operation.
- Drawing of high-color bitmaps

  If the layer works in the same mode as the high-color bitmap has its pixel data available, one DMA2D function call can draw these bitmaps. The following function is used to set up such a function:

  `LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_DRAWBMP_16BPP, pFunc);`
- Filling operations

  Setting up the function for filling operations:

  `LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_FILLRECT, pFunc);`
- Copy operations

  Setting up the functions for copy operations used by the function `GUI_CopyRect()`:
  `LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_COPYRECT, pFunc);`
- Copy buffers

  Setting up the function for transferring the front buffer to the back buffer when using multiple buffers:

  `LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_COPYBUFFER, pFunc);`
- Fading operations

  Setting up the function for mixing up a background and a foreground buffer used for fading memory devices:

  `GUI_SetFuncMixColorsBulk(pFunc);`
- General alpha blending

  The following function replaces the function that is used internally for alpha blending operations during image drawing (PNG or true color bitmaps) or semitransparent memory devices:

  `GUI_SetFuncAlphaBlending(pFunc);`
- Drawing antialiased fonts

  Setting up the function for mixing single foreground and background colors used when drawing transparent ant aliased text:

  `GUI_SetFuncMixColors(pFunc).`

6.5 Hardware JPEG Decoding

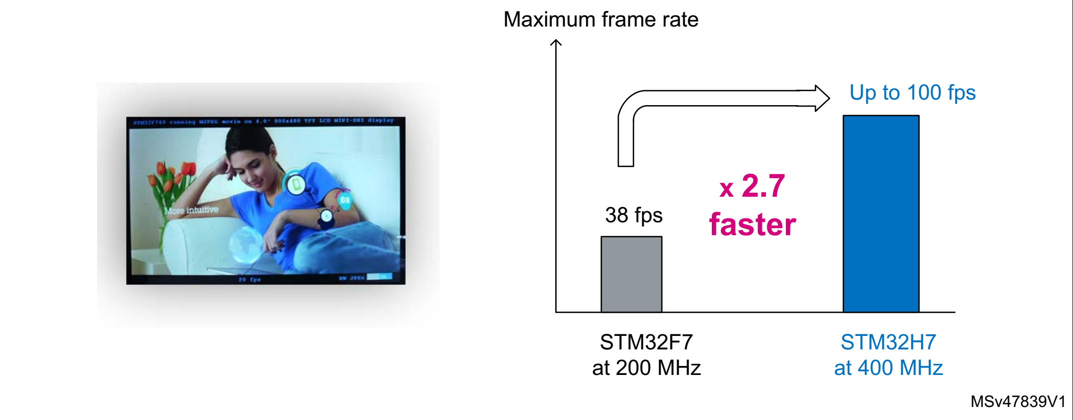
The JPEG peripheral provides a fast and simple hardware compressor and decompressor of JPEG images with the full management of JPEG headers. The JPEG decoder outputs are organized in YCbCr blocks.

The STM32F7 and STM32H7 Series devices support the hardware JPEG decoding.

Using the STM32F7 Series, the conversion from YCbCr to RGB is performed by software. Using the STM32H7 Series, the YCbCr to RGB conversion is accelerated by the Chrom-ART allowing to reach 100 fps with 640x480 resolution versus 38 fps with the STM32F7 Series.

Note: For more details about the JPEG hardware codec performance, please refer to the application note "JPEG hardware codec peripheral in STM32F76/77xxx and STM32H7x3 line microcontrollers" (AN4996).

Figure 22. Hardware JPEG decoding



7 Footprint

The purpose of the following sections is to provide the memory requirements for all the demonstration modules, including JPEG decoder and STemWin main GUI components. The aim is to have an estimation of memory requirement in case of suppression or addition of a module or a feature.

The footprint data is provided for the following environment:

- Toolchain: IAR 7.80
- Optimization: high size
- Board: STM32H743I-EVAL.

Table 6 shows the code memory, data memory and constant memory used for each module.

Table 6. Modules footprint(1)

Module	Code memory (bytes)	Data memory (bytes)	Constant memory (bytes)
Audio	5688	26876	1318945
Video	5424	100228	231664
Graphic effect	2260	112	10041307
Clock & weather	6032	844	3061328
Rocket games	2052	484	1715108
System info	1152	0	120164

1. All the resources used in the demonstration are stored in the Quad-SPI Flash memory.

7.1 STemWin features resources

7.1.1 JPEG decoder

The JPEG decompression uses approximately 33 Kbytes of RAM for decompression independently of the image size, and the number of bytes depends on the image size. The RAM requirement can be calculated as follows:

Approximate RAM requirement = X-Size of image * 80 bytes + 33 Kbytes.

Table 7. RAM requirements for some JPEG resolutions

Resolution	RAM usage (Kbytes)	RAM usage, size-dependent (Kbytes)
160x120	45.5	12.5
320x340	58.0	25.0
480x272	70.5	37.5
640x480	83.0	50.0

The memory required for the decompression is allocated dynamically by the STemWin memory management system. After drawing the JPEG image, the complete RAM is released.

7.1.2 GUI components

The operation area of STemWin varies widely, depending primarily on the application and features used. In the following sections, memory requirements of various modules are listed, as well as the memory requirements of example applications.

Table 8 shows the memory requirements of the STemWin main components. These values depend a lot on the compiler options, the compiler version, and the used CPU. Note that the listed values are the requirements of the basic functions of each module.

Table 8. MemoSTemWin components memory requirements

Component	ROM (Kbytes)	RAM (bytes)	Description
Windows manager	6.2	2.5 K	Additional memory requirements of basic application when using the Windows manager
Memory devices	4.7	7 K	Additional memory requirements of basic application when using memory devices
Antialiasing	4.5	2 * LCD_XSIZE	Additional memory requirements for the antialiasing software item
Driver	2 to 8	20	The memory requirements of the driver depend on the configured driver and the presence of a data cache. With a data cache, the driver requires more RAM.
Multi-layer	2 to 8	-	If the user works with a multi-layer or a multi-display configuration, additional memory is required for each additional layer, as each configuration requires its own driver.
Core	5.2	80	Memory requirements of a typical application without using additional software items
JPEG	12	36 K	Basic routines for drawing JPEG files
GIF	3.3	17 K	Basic routines for drawing GIF files
Sprites	4.7	16	Routines for drawing sprites and cursors
Font	1 to 4		Depends on the font size to be used

Table 9. Widget memory requirements(1)

Component	ROM (Kbytes)	RAM (bytes)
BUTTON	1	40
CHECKBOX	1	52
DROPDOWN	1	52
EDIT	1	28
FRAMEWIN	1	12

Table 9. Widget memory requirements(1) (continued)

Component	ROM (Kbytes)	RAM (bytes)
GRAPH	1	48
GRAPH_DATA_XY	1	-
HEADER	1	32
LISTBOX	1	56
LISTVIEW	1	44
MENU	1	52
MULTIEDIT	1	16
PROGBAR	1	20
RADIO BUTTON	1	32
SCROLLBAR	1	14
SLIDER	1	16
TEXT	1	16
CALENDAR	1	32

1. The listed memory requirements of the widgets contain the basic routines required for creating and drawing the widget. Depending on the specific widget, several additional functions available that are not listed in this table.

8 Functional description of STM32H743I-EVAL, STM32H747I-EVAL, and STM32H747I-DISCO demonstration modules

8.1 STemWin

8.1.1 Audio player

Overview

The audio player module provides a complete audio solution based on the STM32H7 Series 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

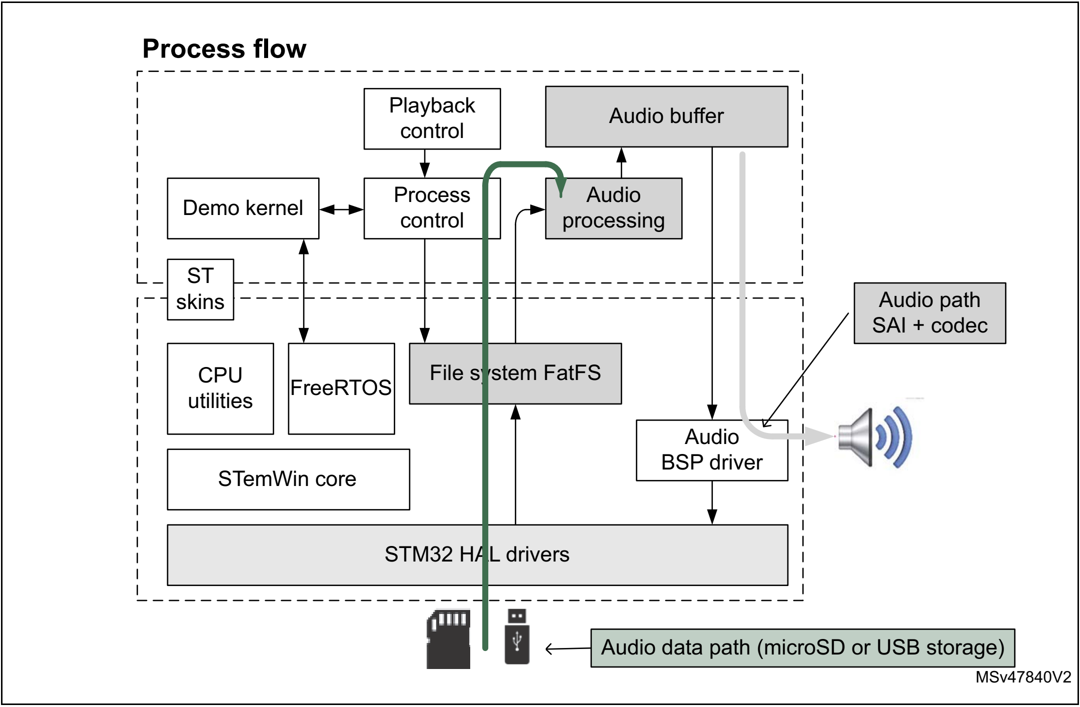
- Audio format: WAV format without compression with 8 k to 96 k sampling
- Embedded equalizer and loudness control
- Performance: MCU Load < 5 %
- Audio files stored in SD card or USB storage
- Only 8-Kbyte RAM required for audio processing

Note: MP3 format supported only by STM32H747 EVAL and Discovery boards.

Architecture

Figure 23 shows the different audio player parts and their connections and interactions with the external components.

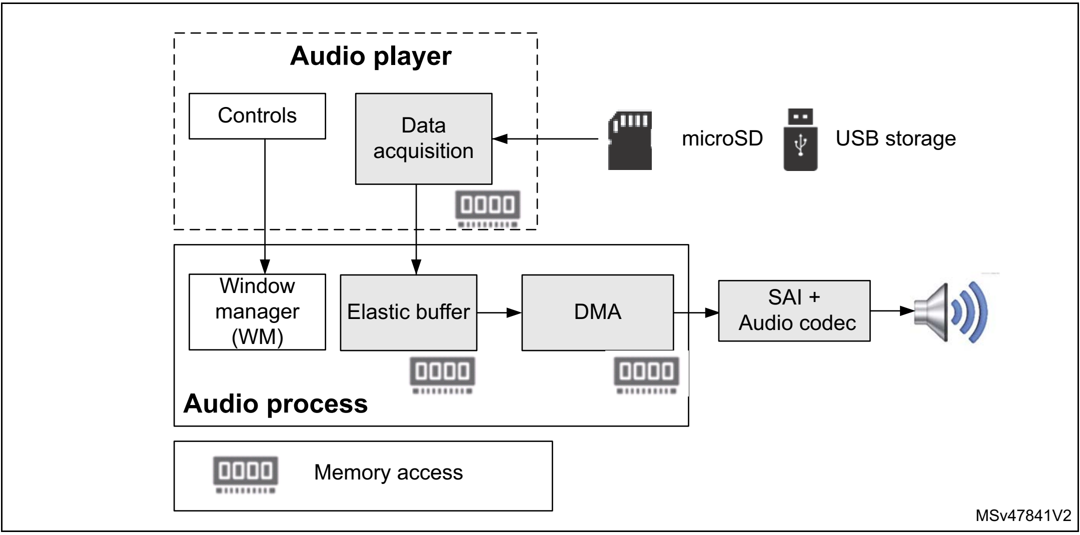
Figure 23. Audio player module architecture



Performance

Figure 24 shows the used performance mechanisms in the audio process and audio player.

Figure 24. Audio player module process

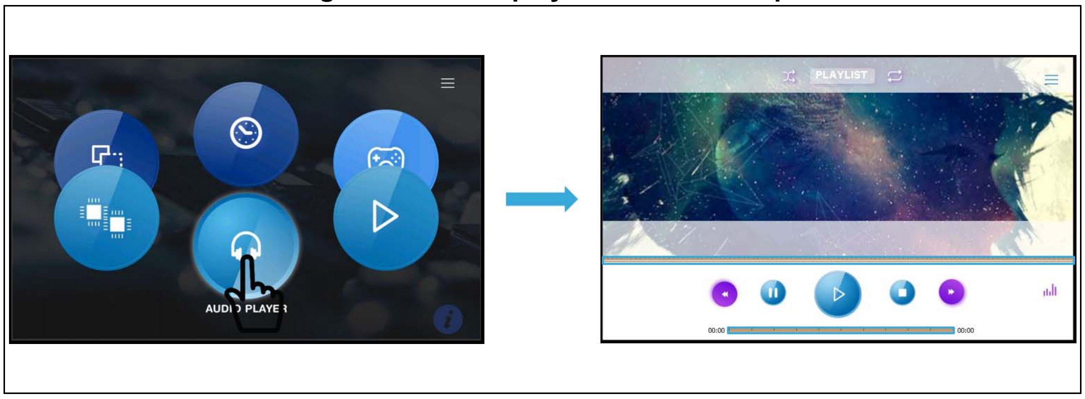


Process description

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 only when the play button in the audio player interface is pressed to start the process.

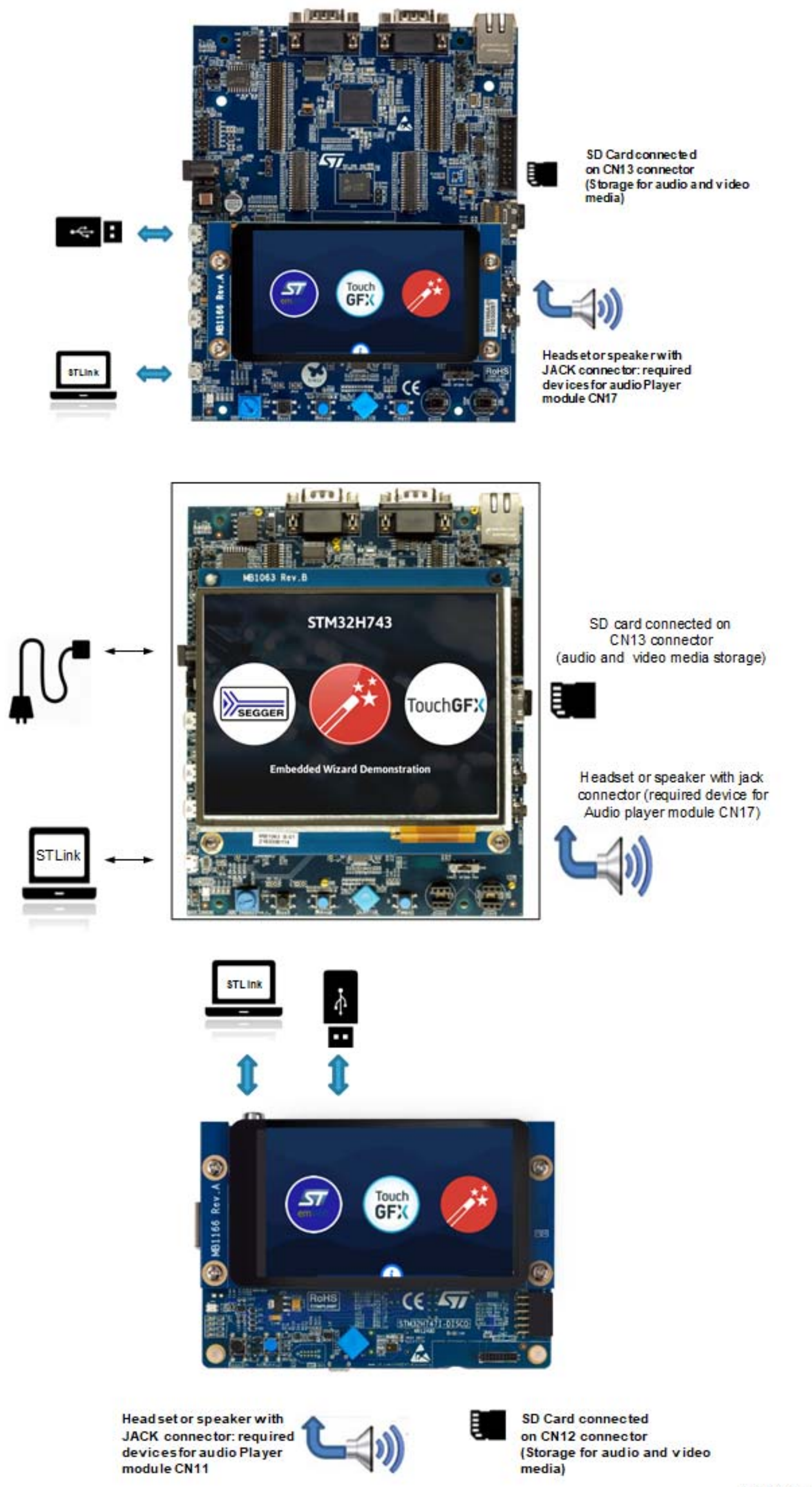
Figure 25 shows the audio player module startup from the main desktop menu.

Figure 25. Audio player module startup



Hardware connectivity

Figure 26. Connectivity of audio player module hardware


SD Card connected on CN13 connector (Storage for audio and video media)
Headset or speaker with JACK connector: required devices for audio Player module CN17
STLink
MB1063 Rev.B
STM32H743
SEGGER
TouchGFX
Embedded Wizard Demonstration
SD card connected on CN13 connector (audio and video media storage)
Headset or speaker with jack connector (required device for Audio player module CN17)
STLink
STLink
Headset or speaker with JACK connector: required devices for audio Player module CN11
SD Card connected on CN12 connector (Storage for audio and video media)
MSv47845V2

Audio player module controls

Table 10. Audio player module controls

Button	Preview	Brief description
Play	▶	– Change the audio player state to AUDIOPLAYER_PLAY – Read the wave file from the storage unit – Set the frequency – Start or resume the audio task – Start playing an audio stream from a data buffer using BSP_AUDIO_OUT_Play function in BSP audio driver – Replace the play button by pause button
Pause	⏸	– Suspend the audio task – Pause the audio file stream – Replace the pause button by play button
Stop	⏹	– Close the wave file from the storage unit – Suspend the audio task – Stop audio playing – Change the audio player state to AUDIOPLAYER_STOP
Previous	⏪	– Point to the previous wave file – Stop audio playing – Start playing the previous wave file if the play button is pressed
Next	⏩	– Point to the next wave file – Stop audio playing – Start playing the next wave file if the play button is pressed
Add file to playlist	PLAYLIST	– Open playlist window – Add an audio file from SD card or choose an audio file to play
Visualization	📊	Disable or enable the display of graphic animation based on real time PCM audio samples
Menu	☰	Close the audio player module
Volume	―	– Volume up – Volume Down
Time	00:00 ― 00:00	– Move the Music time forward – Move the Music time back

8.1.2 Video player

Overview

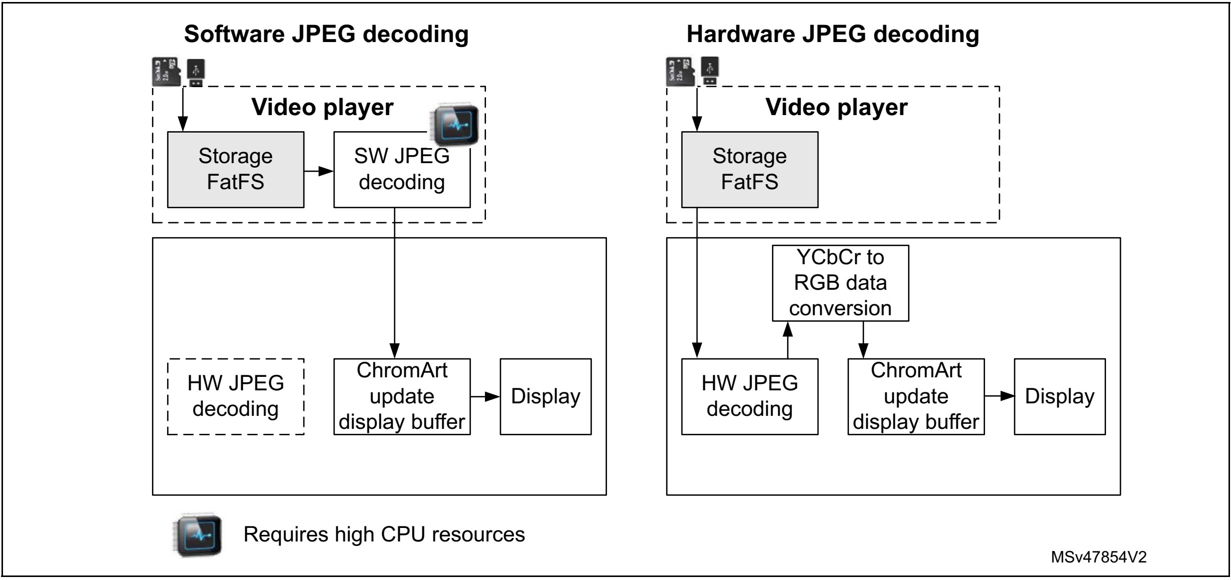
The video player module provides a video solution based on the STM32H7 Series and STemWin movie API. It supports playing a movie in AVI format.

Features

- Video format: AVI video format
- Performance: MCU Load < 5 % and rate up to 25 fps
- Video files stored in SD card
- Use of the two LCD layers (playback control and video display)
- 64-Kbyte RAM required for JPEG decoding

Figure 27 shows the different video player modules and their connections and interactions with the external components.

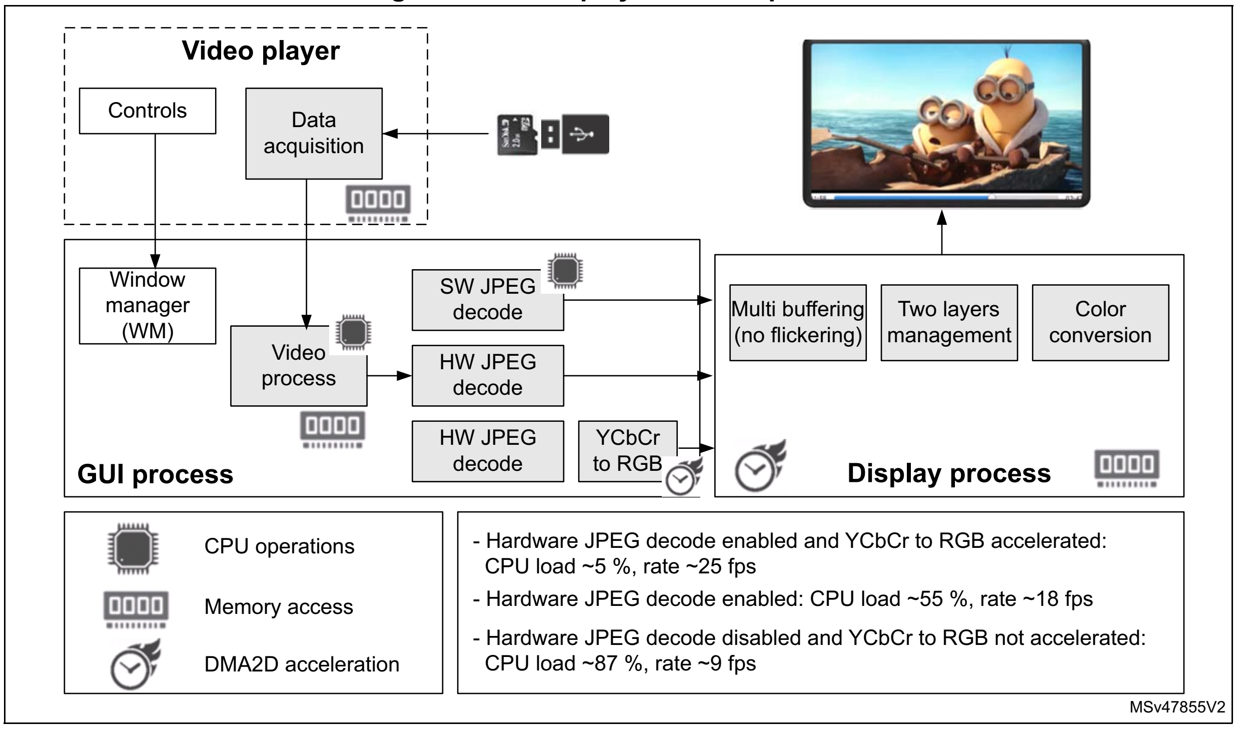
Figure 27. Video player module architecture



Performance

Figure 28 shows the GUI, display and video player process and performance.

Figure 28. Video player module process

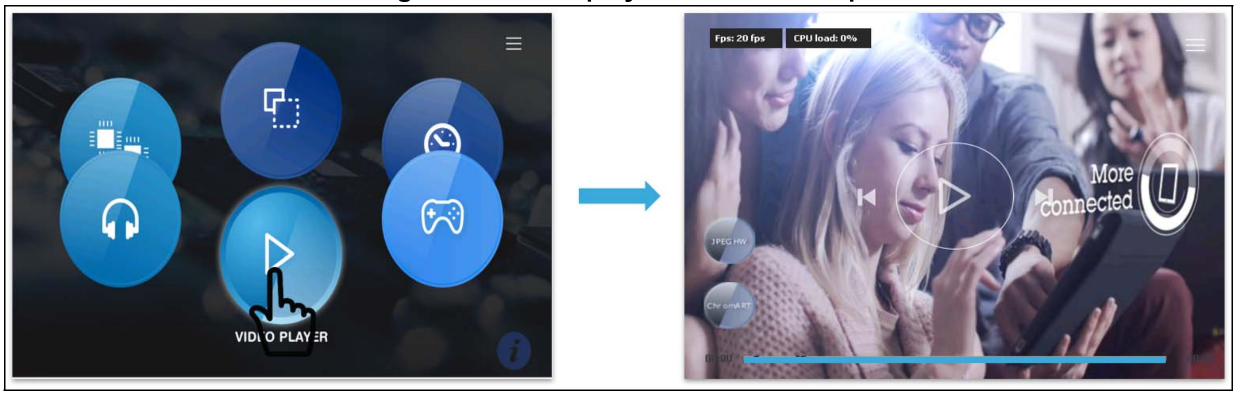


- Hardware JPEG decode enabled and YCbCr to RGB accelerated: CPU load ~5 %, rate ~25 fps
- Hardware JPEG decode enabled: CPU load ~55 %, rate ~18 fps
- Hardware JPEG decode disabled and YCbCr to RGB not accelerated: CPU load ~87 %, rate ~9 fps

Functional description

Figure 29 shows the video player module startup by touching the video player icon.

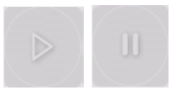
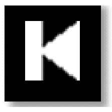
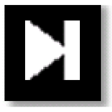
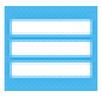
Figure 29. Video player module startup



Note: After 5 seconds without touching the screen, the video controls buttons Play, Next, Previous and Exit disappear.

Table 11 summarizes the different actions behind each control button.

Table 11. Video player module controls

Button	Preview	Brief description
Play Pause		– Read the AVI file from the storage unit – Start playing an audio stream – Replace the play/pause button by pause/play button
Previous		– Point to the previous AVI file – Stop video playing – Start playing the previous AVI file
Next		– Point to the previous AVI file – Stop video playing – Start playing the previous AVI file
HW JPEG		– Enable and disable the JPEG hardware decoding
ChromART		– Enable and disable the use of Chrom-ART for YCbCr to ARGB conversion
Exit		– Close video player module
Volume		– Volume up – Volume down
		– Move the video time forward – Move the video time back

8.1.3 Rocket game

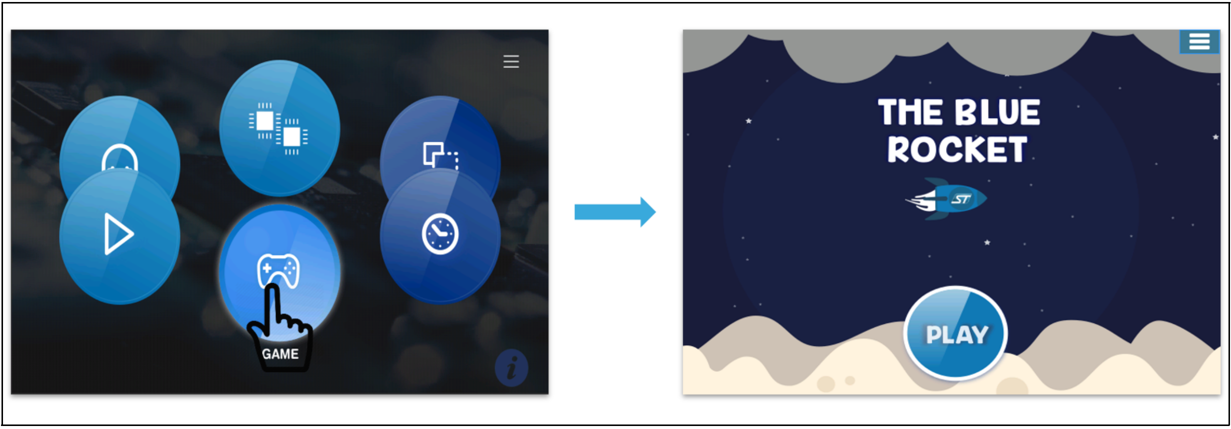
Overview

The rocket game shows the graphic performance of the Chrom-ART Accelerator. The objective is to control the rocket by moving it on the screen. The player has to collect the maximum number of coins to get the best score.

Functional description

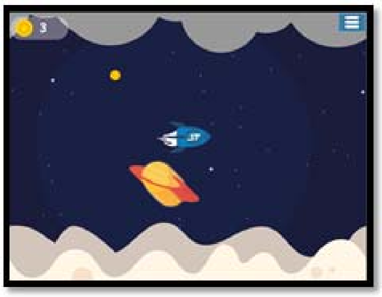
1. Start the rocket game by touching the game icon (see *Figure 30*).

Figure 30. Rocket game startup



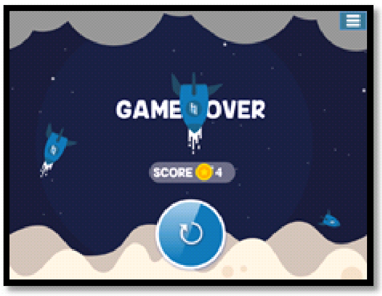
2. Press the play button to start playing and control the rocket by moving on the screen to collect the maximum number of coins and avoid the crash with the planets.

Figure 31. Rocket game ongoing



3. Game is over when the rocket crashes the planet.

Figure 32. Rocket game end



8.1.4 Clock and weather

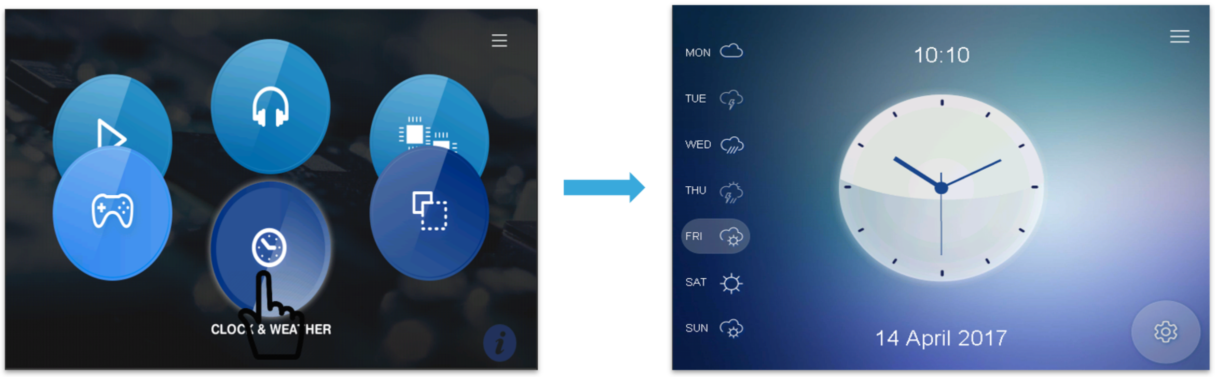
Overview

The clock and weather module allows the time and date display and adjustment by changing the real-time configuration (RTC).

Note: Only the graphical aspect of the weather functionality is integrated.

1. Start the module by touching the clock and weather icon.

Figure 33. Clock and weather startup



2. Press the Settings button to choose the clock and change the skin.

Figure 34. Clock and weather module settings

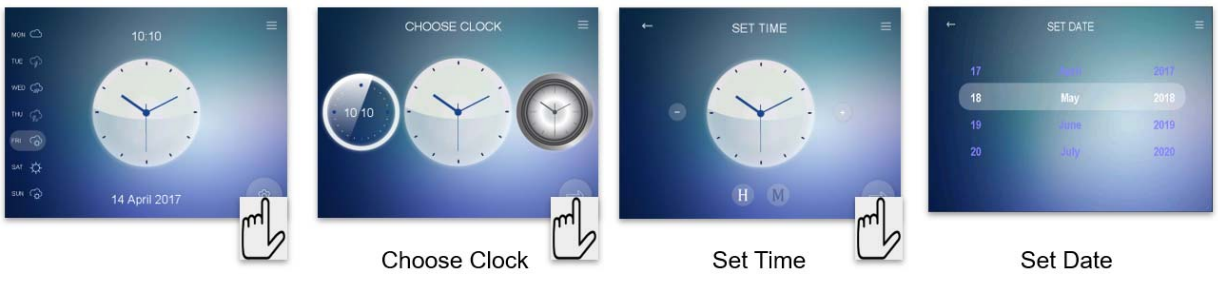
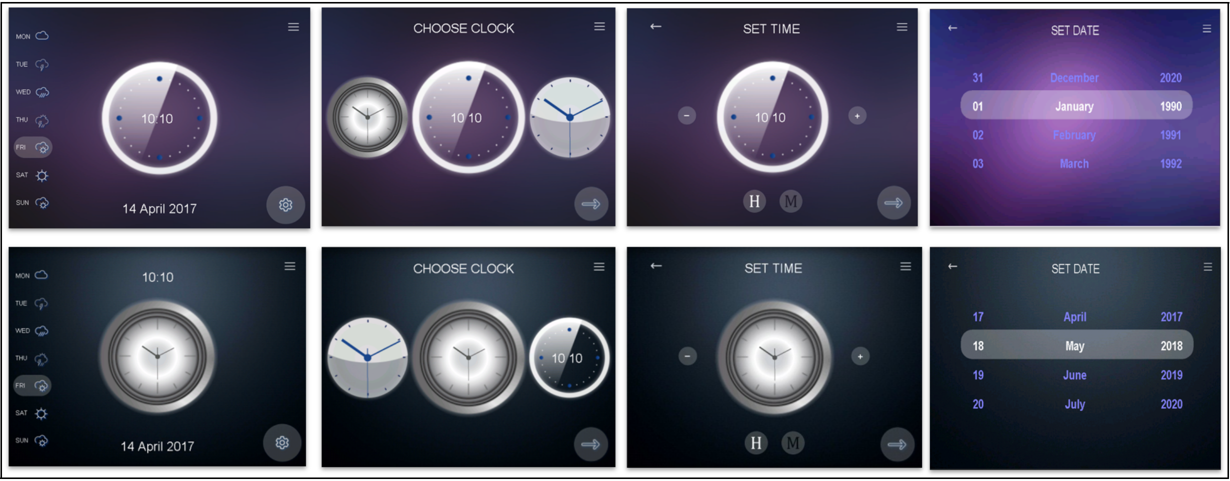


Figure 35 shows the different skins available when changing the clock.

Figure 35. Clock and weather module skins



3. Press Next and Previous buttons to set the time and date.
4. Press the Menu button to return to the main window of the module.

8.1.5 Graphic effect

Overview

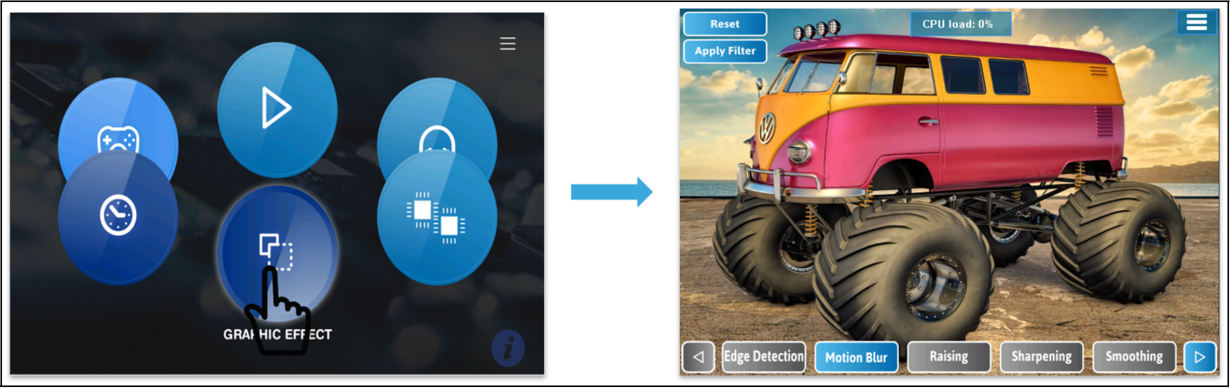
The graphic effect module demonstrates the computing capabilities of the platform to render a real-time effect at full-screen resolution.

The implemented filters are the following:

- Edge detection filter
- Smoothing filter
- Sharping filter
- Raising filter
- Motion blur filter

The CPU load metrics are displayed in the middle of the top screen.

Figure 36. Graphic effect main screen



Note: The BMP files used with this module need to be stored in the SD card or USB storage and under the BMP folder.

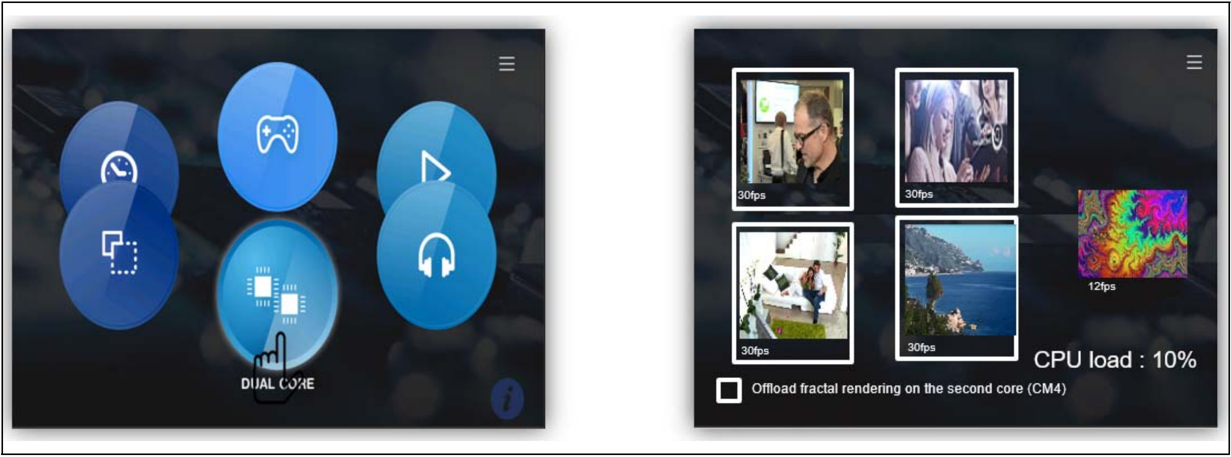
8.1.6 Dual-core module

Overview

The dual-core module demonstrates the capability of the dual-core platforms while decoding four video files and rendering fractal bitmap.

Checking the checkbox offloads the fractal rendering on the second core (CM4). This must decrease the CM7 core loading.

Figure 37. Dual-core module

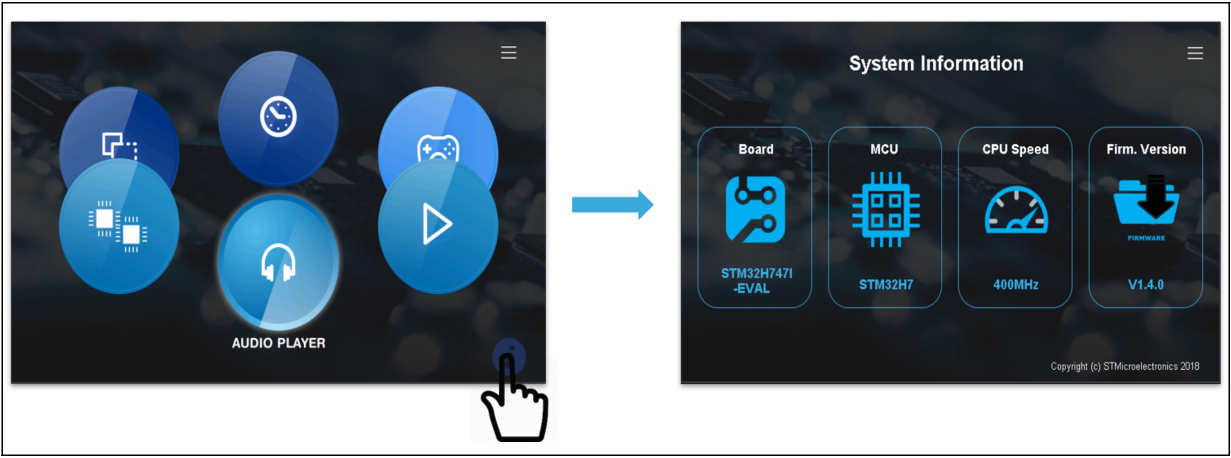


8.1.7 System information

Overview

The system information shows the main demonstration information, such as the used board, the STM32H7 part number, the current CPU clock, and the demonstration revision.

Figure 38. System information main screen



8.2 TouchGFX demonstration

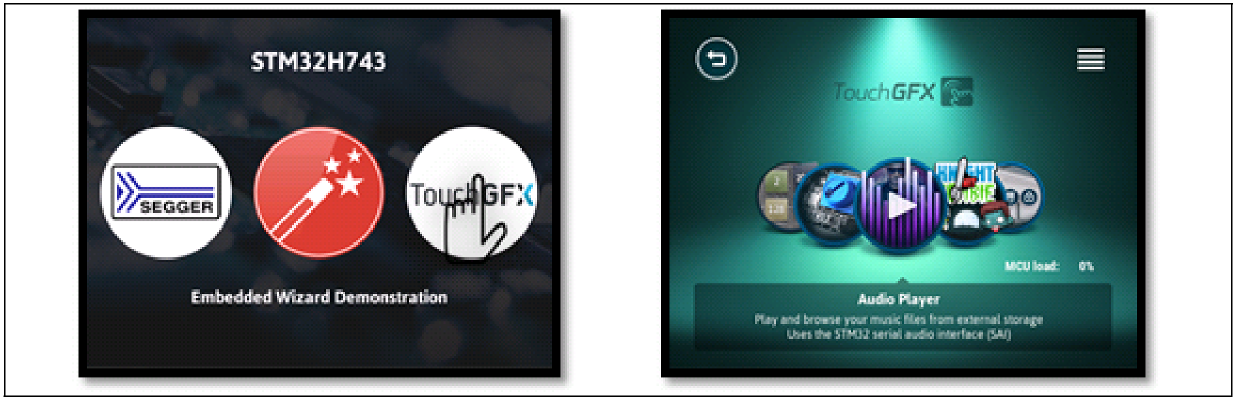
Overview

The TouchGFX demonstration is available in binary format.

To show the TouchGFX demonstration, the user needs to load the full binary file available under Demonstration/binaries

- STM32CubeDemo_STM32H743-Eval_VX.Y.Z_FULL.hex.

Figure 39. TouchGFX demonstration


STM32H743
SEGGER
TouchGFX
Embedded Wizard Demonstration
TouchGFX
MCU load: 0%
Audio Player
Play and browse your music files from external storage
Uses the STM32 serial audio interface (SAI)

8.2.1 Audio player module

Overview

The audio player module provides a complete audio solution based on the STM32H7 Series 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. In the audio module, the user can select which song to play, whether by selecting it from the playlist consisting of all the songs on the device or by going through the directories where the songs are. The audio player also consists of an equalizer that enables the user to adjust the sound.

Figure 40. TouchGFX - Audio player module

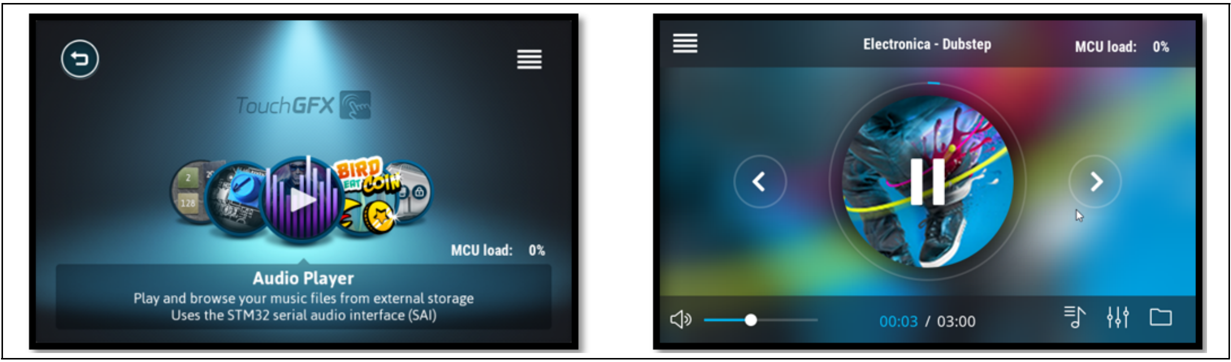
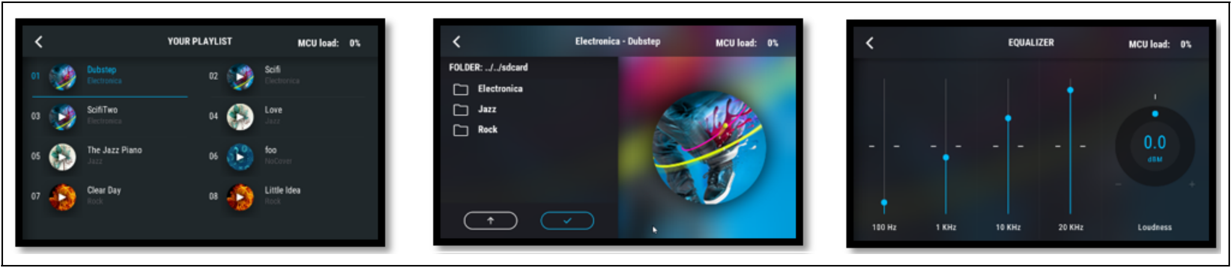

TouchGFX
MCU load: 0%
Audio Player
Play and browse your music files from external storage
Uses the STM32 serial audio interface (SAI)
Electronica - Dubstep
MCU load: 0%
00:03 / 03:00

Figure 41. The three audio menus, playlist (left), directories (middle) and equalizer (right)



8.2.2 Graphics effect

Overview

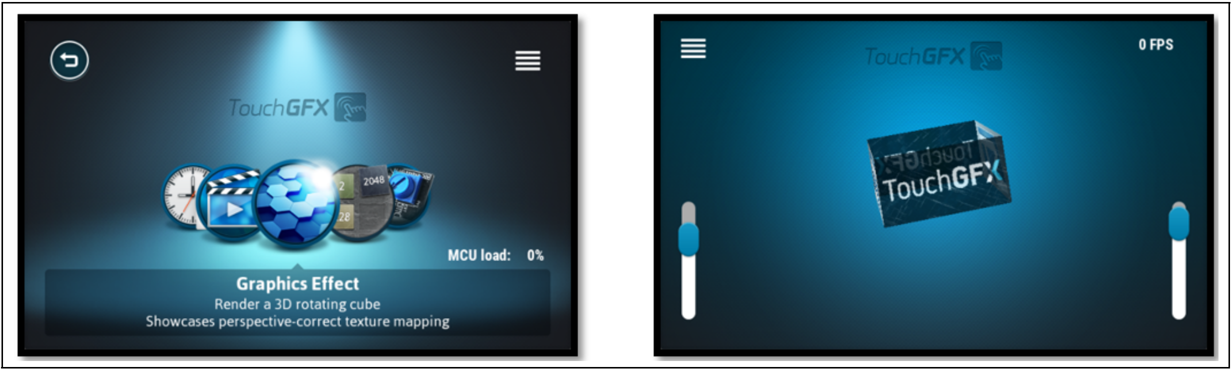
Graphics Effect shows how TouchGFX is able to use the texture mapper to change the transparency, rotate and scale images at run-time.

The module consists of a 3D cube, which rotates, and two sliders, with which the user can change the scale and transparency of the cube.

Functional Description

1. When the module is entered, a cube is visible in the center of the screen.

Figure 42. Graphics effect module


TouchGFX
MCU load: 0%
Graphics Effect
Render a 3D rotating cube
Showcases perspective-correct texture mapping
0 FPS
TouchGFX

2. Moving the left slider changes the transparency and moving the right slider, changes the scale.
3. Swiping the cube results in the rotation of the cube speeds up in the direction of the swipe.

8.2.3 Video player module

Overview

The video player module provides a video solution based on the STM32H7 Series and the TouchGFX APIs. It supports the playing movie in AVI format.

In the video player module, the user is able to see the effect of the JPEG decoder, by turning it on/off while a video is playing. Selecting a video is whether done via a playlist consisting of all the videos on the device, or by going through the directories where the videos are placed.

Figure 43. TouchGFX - Video player module

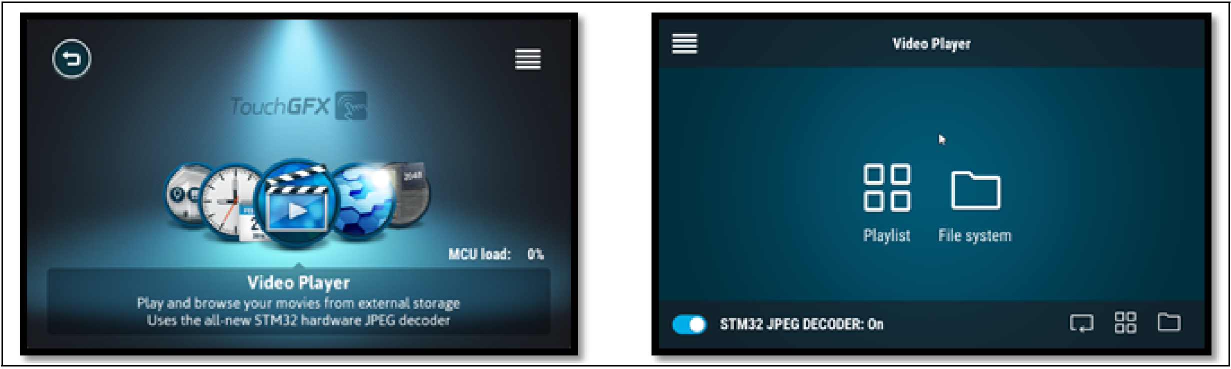
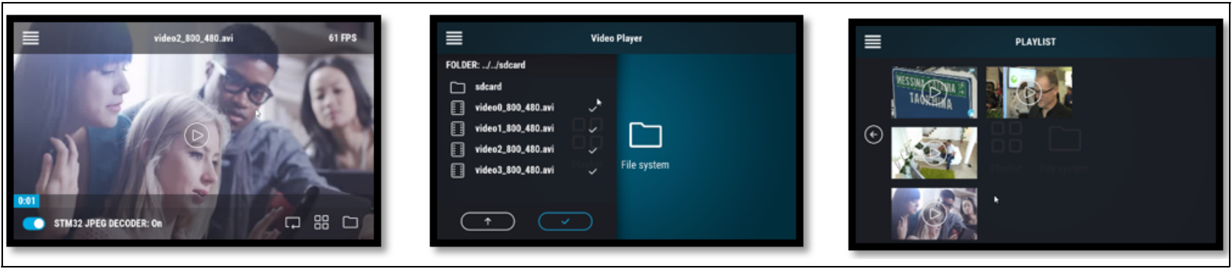

TouchGFX
MCU load: 0%
Video Player
Play and browse your movies from external storage
Uses the all-new STM32 hardware JPEG decoder
Video Player
Playlist
File system
STM32 JPEG DECODER: On

Figure 44. The three video player menus, video player (left), directory (middle) and playlist (right)



8.2.4 Time and calendar module

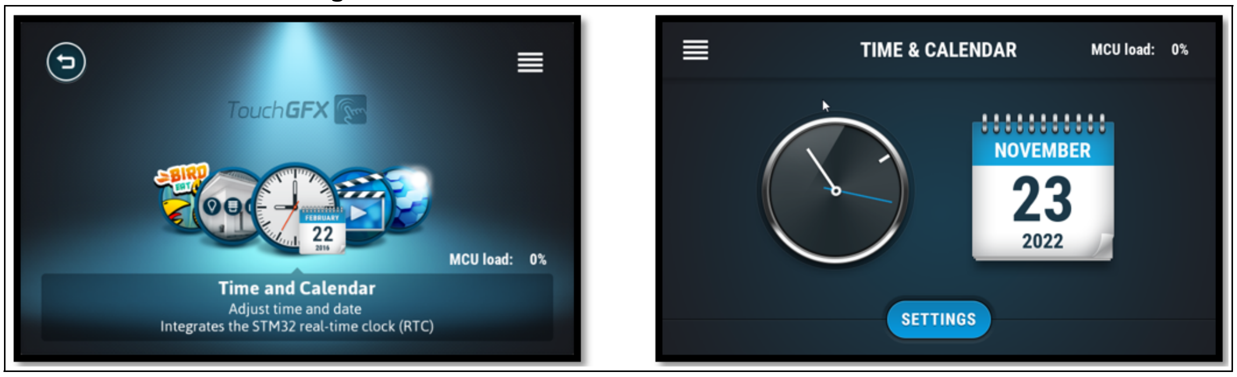
Overview

The time and calendar let the user change the time and date which is handled by the real-time configuration (RTC). In the module, the user is also able to select different watch faces to show the time.

Functional description

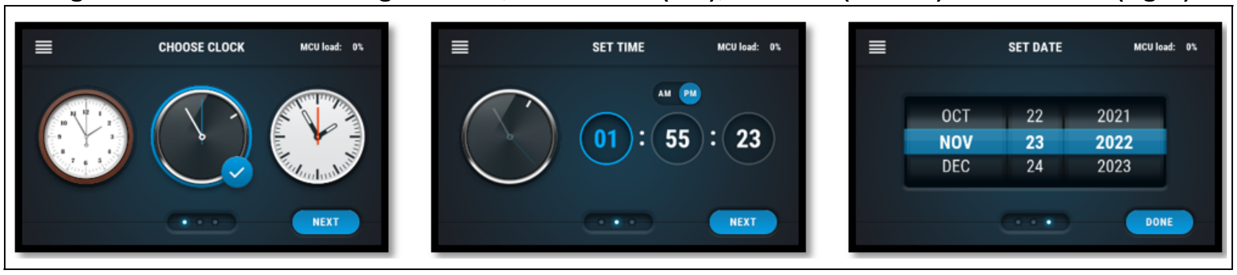
1. Entering the module, a clock and a calendar display the time that is currently in the RTC, together with the selected watch face.

Figure 45. TouchGFX - Time and calendar module



2. Pressing the settings button changes the menu to the clock face setting, which is selected by pressing next.
3. After choosing a clock face, the user can set the time and confirm it by pressing next.
4. The last setting is the date, which is set by pressing done. If the icon in the top left corner of the Time and calendar module is exited, the changes that are not confirmed by pressing done are discarded.

Figure 46. The three settings menus, watch face (left), set time (middle) and set date (right)



8.2.5 Home control module

Overview

The Home Control module allows the user to control the lights, blinds, and security for the different rooms in a house. The module also lets the user view statistic based on the controllable elements.

Functional description

1. Entering the module brings up a menu, which lets the user select between the four options Light, Blinds, Security and Statistics.

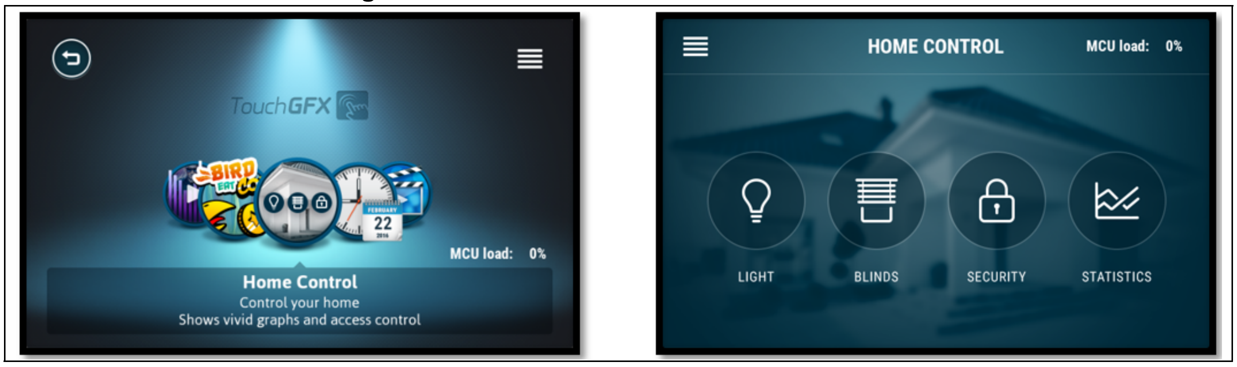
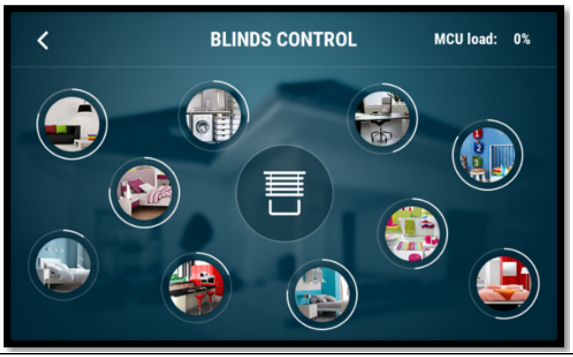


Figure 47. TouchGFX - Home control module

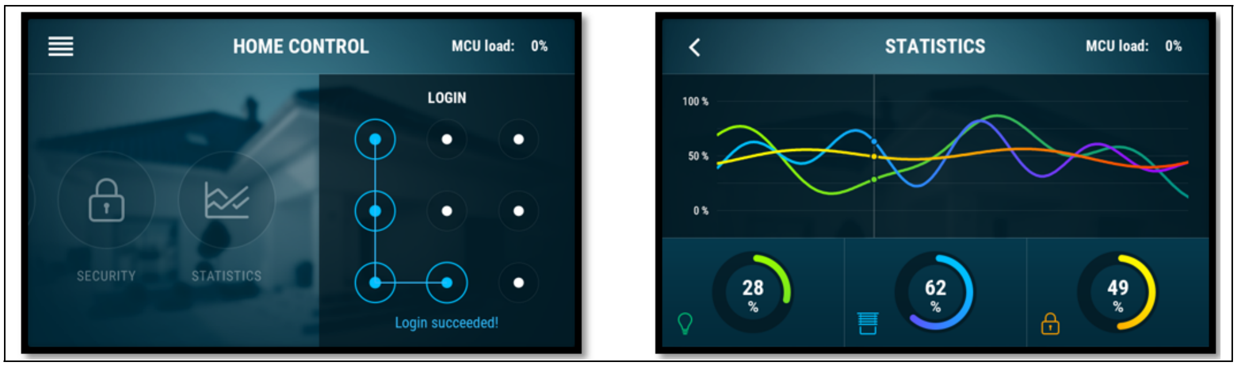
2. In the light and security menus, the user is able to turn the two settings ON/OFF for single rooms and in the blinds menu, the user is able to adjust how much the blinds are open.

Figure 48. Menu to set the blinds, like light and security but options only to tune on and off



3. Entering the security and statistics menus requires a login in the pattern.
4. In the statistics screen, graphs are used to show the statistics for the three options.

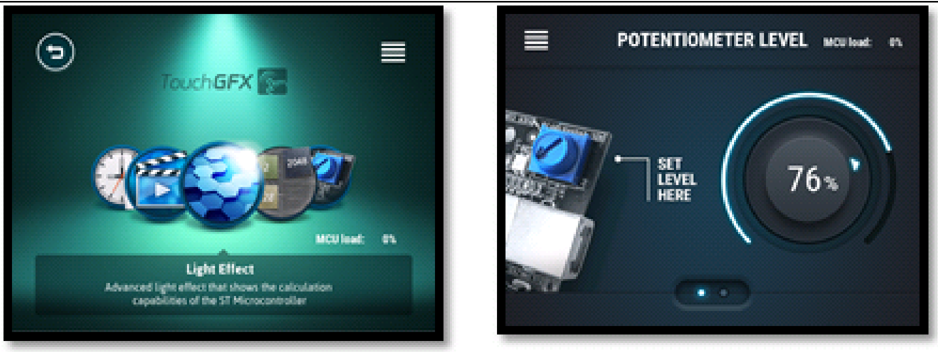
Screen lock (left) and statistics menu (right)



8.2.6 Light effect module

The light effect module shows the calculation capabilities of the STM32H7 microcontroller via an advanced controlled light effect.

Figure 49. TouchGFX - Light effect module



Advanced light effect that shows the calculation capabilities of the ST Microcontroller

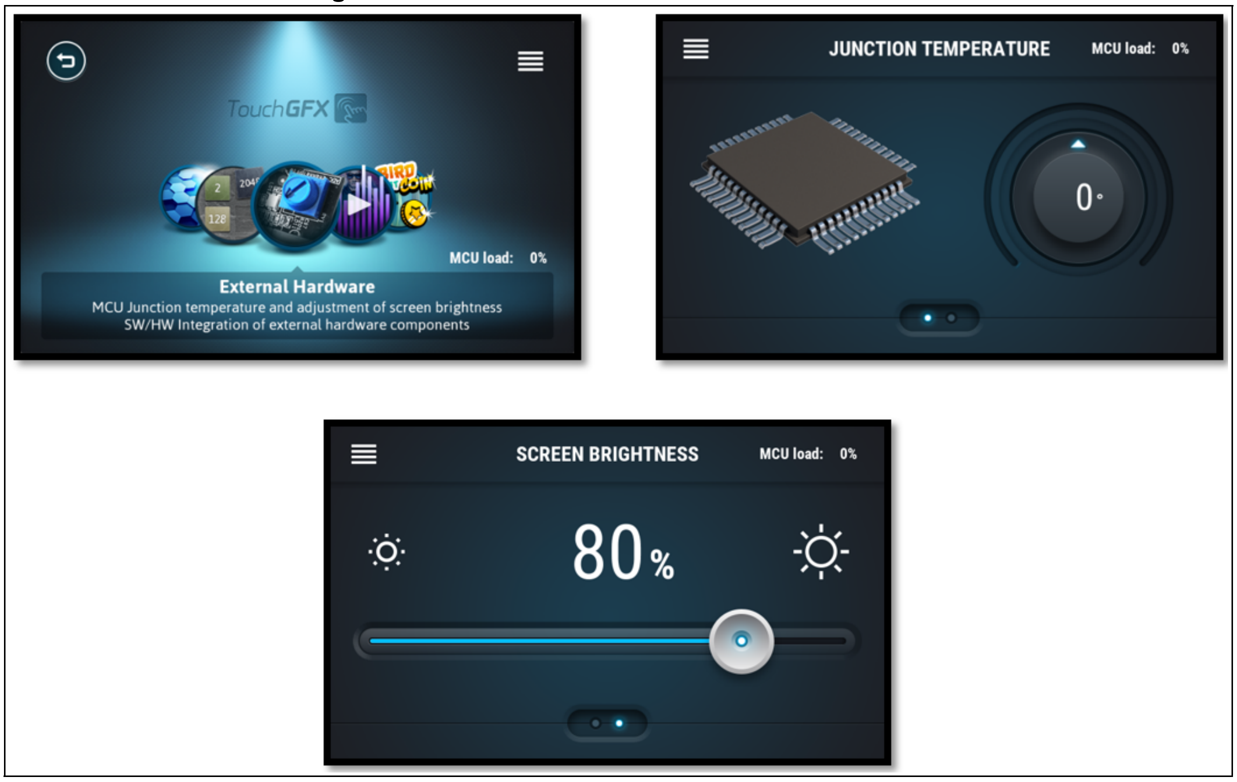
8.2.7 External hardware module

The external hardware module shows how TouchGFX can communicate with other parts of the H7.

In the module, the user is able to view the MCU Junction temperature and adjust the screen brightness.

When entering the module the user is presented with the Junction temperature and by swiping right, the user is able to use a slider to adjust the screen brightness.

Figure 50. TouchGFX - External hardware module


TouchGFX
MCU load: 0%
External Hardware
MCU Junction temperature and adjustment of screen brightness
SW/HW Integration of external hardware components
JUNCTION TEMPERATURE
MCU load: 0%
0°
SCREEN BRIGHTNESS
MCU load: 0%
80%

8.2.8 Bird Eat Coin

Overview

The Bird Eat Coin game highlights the graphic performance of the Chrom-ART Accelerator.

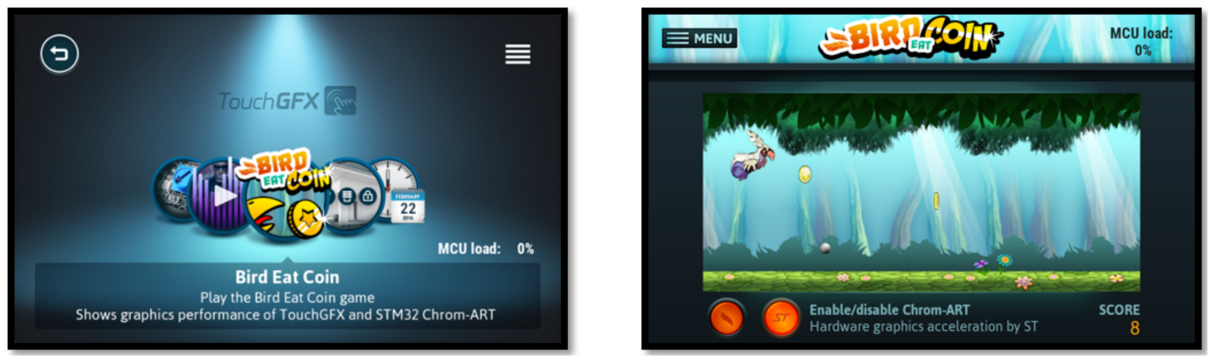
The goal of the game is to use the bird to ‘eat’ the coins coming at the bird to score points.

If the bird eats a bullet instead of a coin the game is lost and the score resets to 0.

Functional Description

1. When entering the Bird Eat Coin from the main menu the game starts.

Figure 51. Bird Eat Coin game

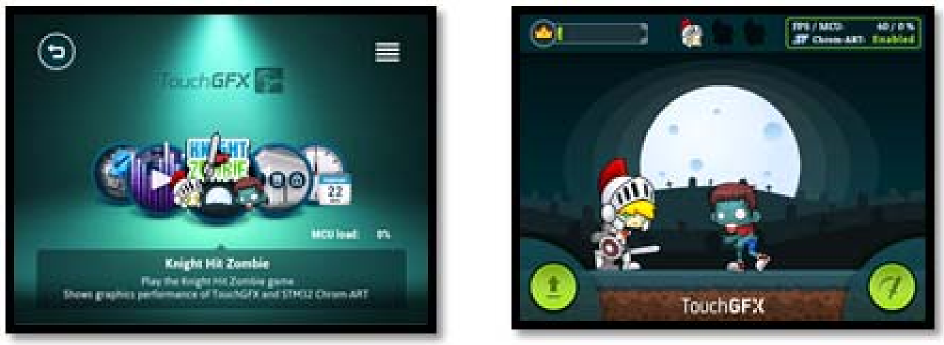

TouchGFX
MCU load: 0%
Bird Eat Coin
Play the Bird Eat Coin game
Shows graphics performance of TouchGFX and STM32 Chrom-ART
MENU
BIRD EAT COIN
MCU load: 0%
Enable/disable Chrom-ART
Hardware graphics acceleration by ST
SCORE
8

2. To move the bird up and down, the user has to press the left button, with the wing, in the bottom to move up, and release it to move down.
3. To see the performance enhancement that the Chrom-ART Accelerator delivers, Chrom-ART can be enabled or disabled by pressing the right button, with the ST logo, at the bottom.

8.2.9 Knight Hit Zombie game

The Knight Hit Zombie game shows the graphic performance of the Chrom-ART Accelerator and TouchGFX graphical stack.

Figure 52. TouchGFX - Knight Hit Zombie game


TouchGFX
MCU load: 0%
Knight Hit Zombie
Play the Knight Hit Zombie game
Shows graphics performance of TouchGFX and STM32 Chrom-ART
FPS / MCU: 60 / 0 %
Chrom-ART: Enabled
TouchGFX

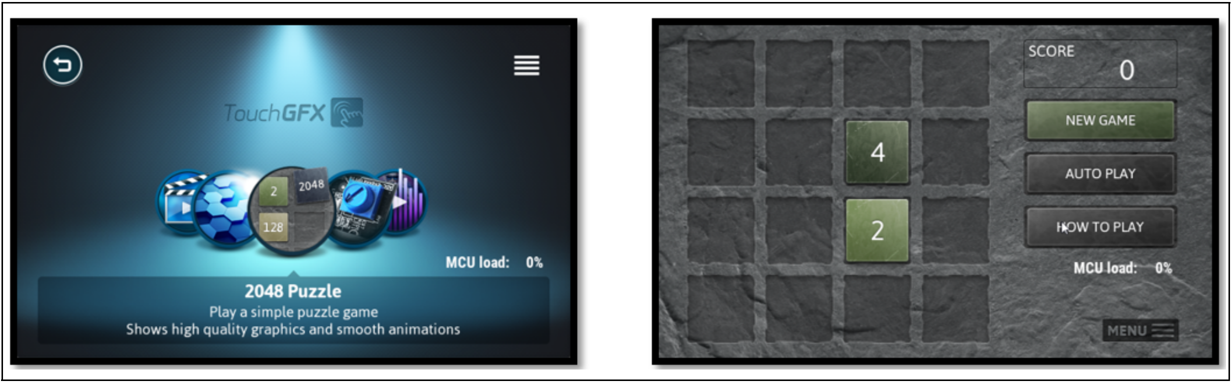
8.2.10 2048 Puzzle game

Overview

The 2048 game module, is the classic game 2048 created with TouchGFX, which showcases the high-quality graphics and smooth animations that TouchGFX delivers. In the game, the user can either play the game or simulate a game by selecting `Auto Play`. To describe how the game is played a `How To Play` button opens a modal window, which contains the rules.

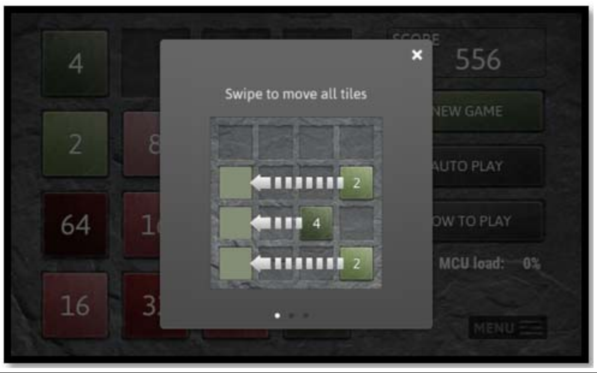
1. When the 2048 module is entered, the game is started and the user can interact with the game board.

Figure 53. 2048 Puzzle game module



2. To restart the game the user can select the button New Game.
3. Simulating a game is done by pressing the Auto Play button and the tiles I being moved by the H7.
4. Selecting `How To Play` opens a modal window, which describes the rules and the goal of the game.

Figure 54. The *How To Play* modal window



8.3 Embedded wizard demonstration

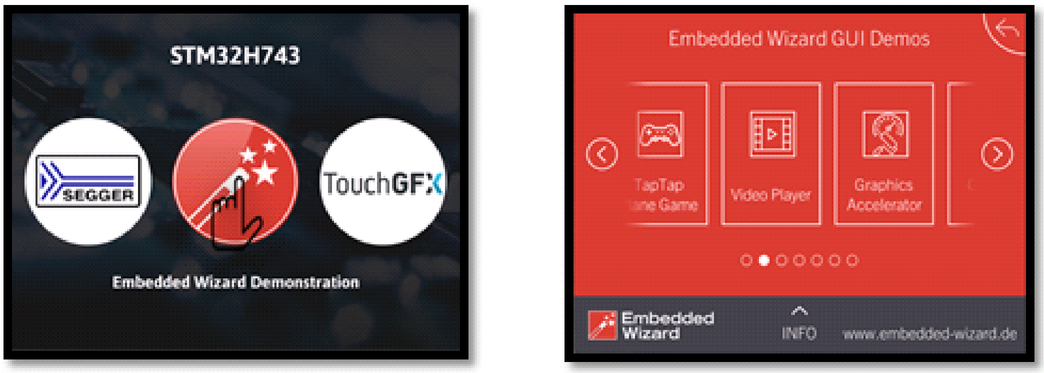
8.3.1 Overview

The embedded wizard demonstration is available in binary format.

To show the embedded wizard demonstration, the user needs to load the full binary file available under *Demonstration/binaries*

- STM32CubeDemo_STM32H743-Eval_VX.Y.Z_FULL.hex

Figure 55. Embedded wizard demonstration



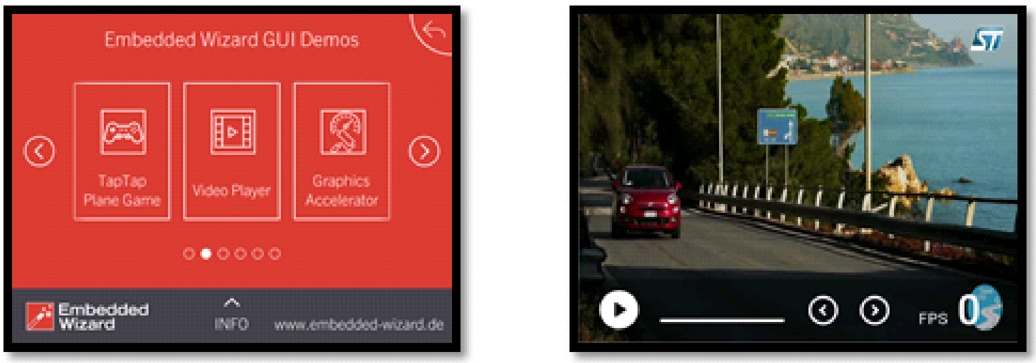
8.3.2 Video player module (only H743I-EVAL)

The video player module provides a video solution based on the STM32H7 Series and the embedded wizard APIs. It supports the playing movie in AVI format.

The video files must be named as follows:

- video0_800_480.avi
- video1_800_480.avi
- video2_800_480.avi
- video3_800_480.avi

Figure 56. Embedded wizard - Video player module



8.3.3 Graphic effect module (only H743I-EVAL)

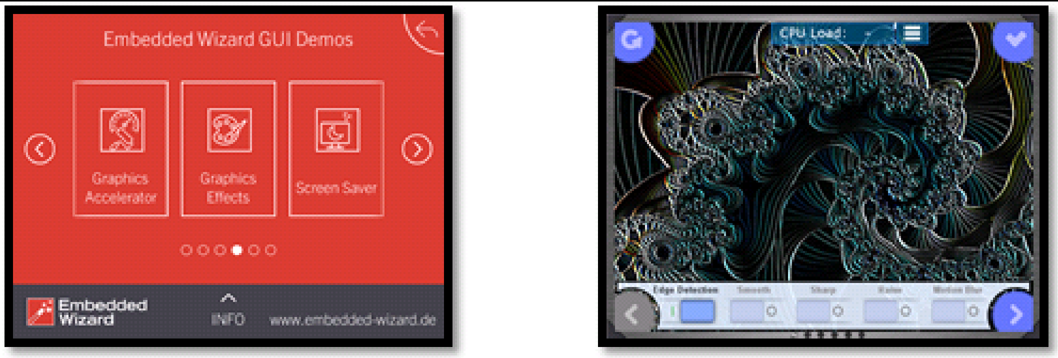
The graphic effect module demonstrates the computing capabilities of the platform to render a real-time effect at full-screen resolution.

The implemented filters are the following:

- Edge detection filter
- Smoothing filter
- Sharping filter
- Raising filter
- Motion blur filter

The CPU load metrics are displayed in the middle of the top screen.

Figure 57. Embedded wizard - Graphic effect module

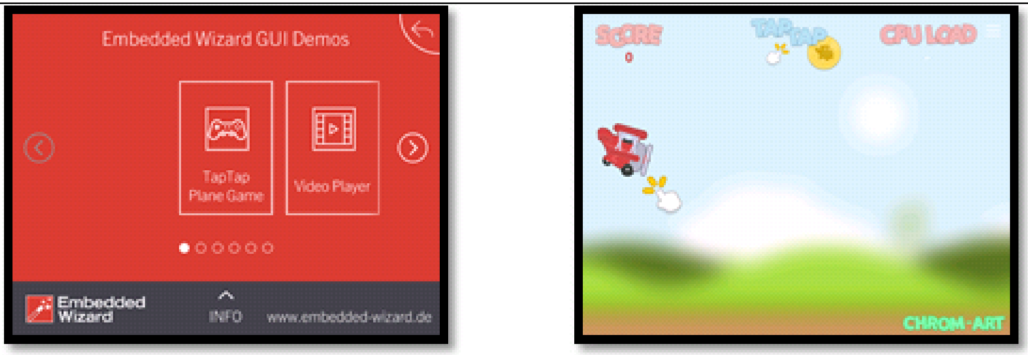


8.3.4 TapTap plane module (only H743I-EVAL)

The TapTap plane module allows the user to control a plane by touching anywhere on the screen. The player has to collect the maximum number of stars to get the best score.

To display the processing capabilities of the STM32H7 Series, the user can enable or disable the Chrom-ART Accelerator by pressing on the Chrom-ART button at the bottom right of the screen.

Figure 58. Embedded wizard - TapTap plane module



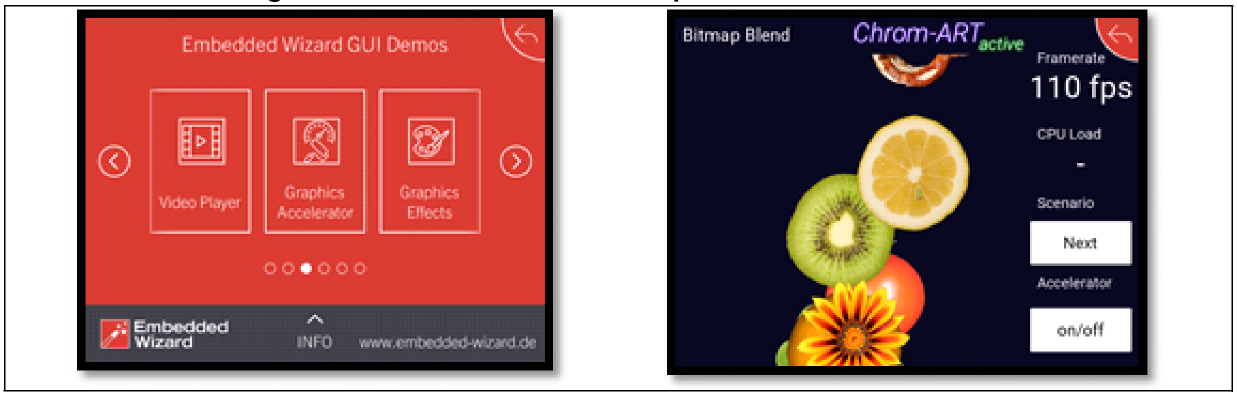
8.3.5 Graphics accelerator module

The graphic accelerator module is used to display the Chrom-ART accelerator capabilities and how it offloads the CPU.

The five operations listed below are used:

- Alpha8 blend
- Rectangle copy
- Bitmap copy
- Rectangle blend
- Bitmap blend

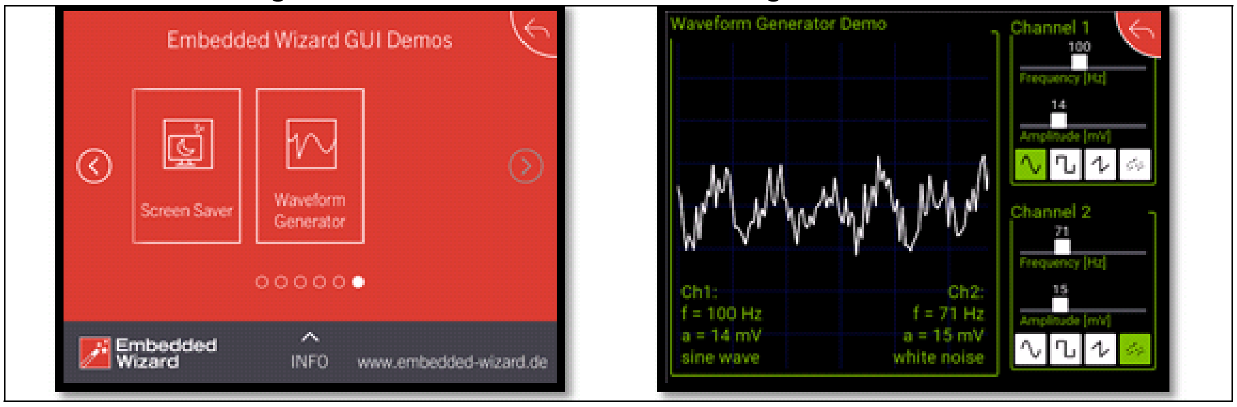
Figure 59. Embedded wizard - Graphics accelerator module



8.3.6 Waveform generator module

The waveform generator module displays the possibility to emulate waveform generation and signal frequency display.

Figure 60. Embedded wizard - Waveform generator module

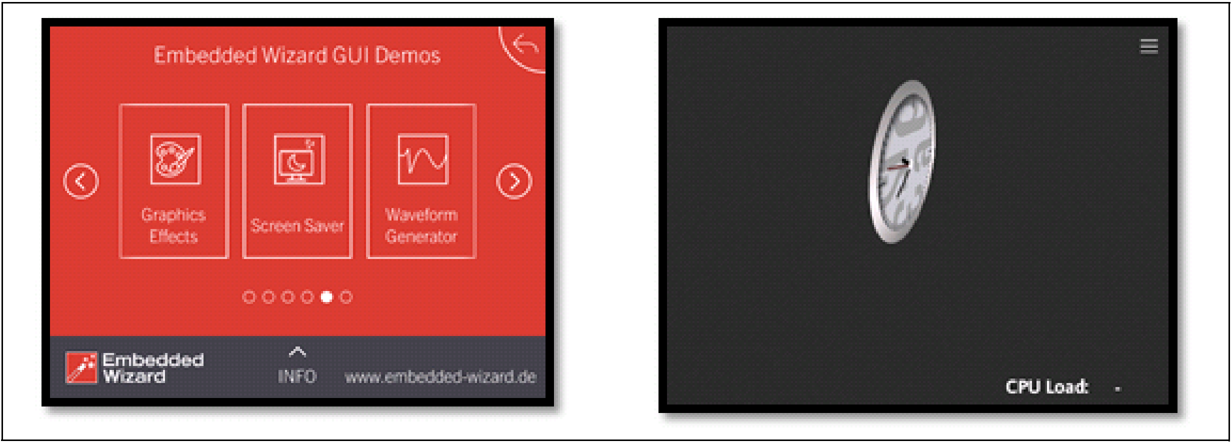


8.3.7 Screen saver module (only H743I-EVAL)

The screen saver module is used to display a rotating clock that provides the current time while it is spinning around the three axes (x, y, and z).

The spinning operation uses intensive CPU calculation to highlight the CPU capabilities.

Figure 61. Embedded wizard - Screen saver module

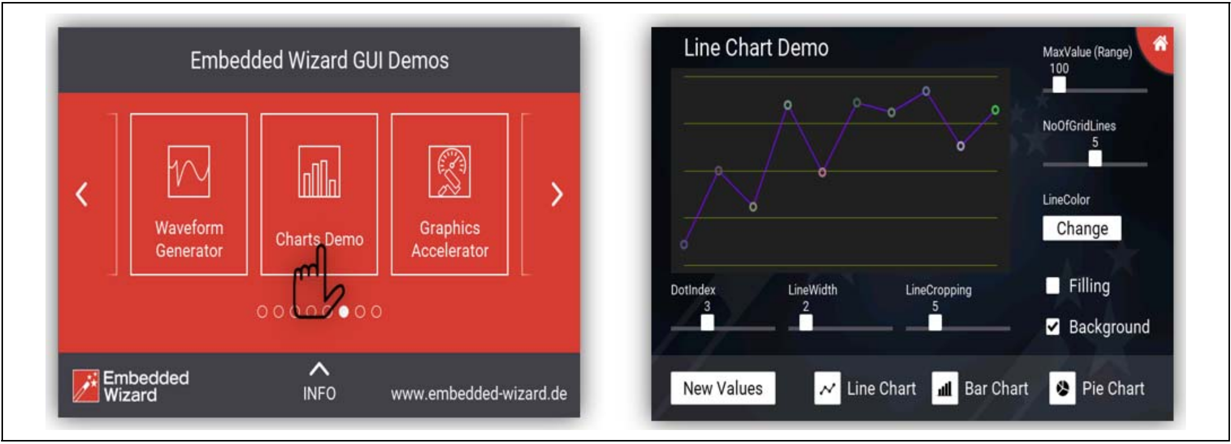


8.3.8 Charts demonstration

Overview

Graphic module showing charts demonstration.

Figure 62. Charts demonstration

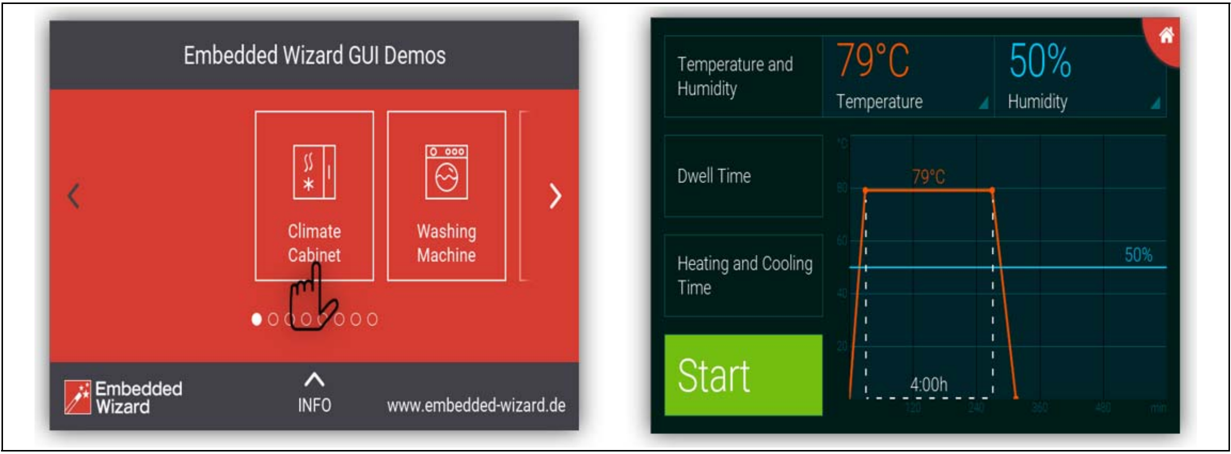


8.3.9 Climate cabinet

Overview

Graphic module showing a climate cabinet application.

Figure 63. Climate cabinet

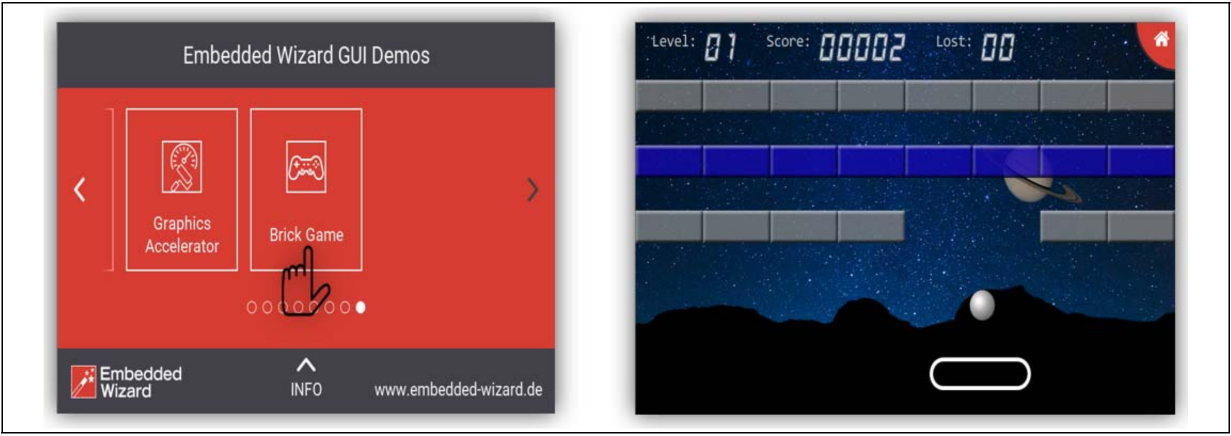


8.3.10 Brick game

Overview

Control the paddle by touching and moving it right and left to bounce the ball and destroy all bricks.

Figure 64. Brick game

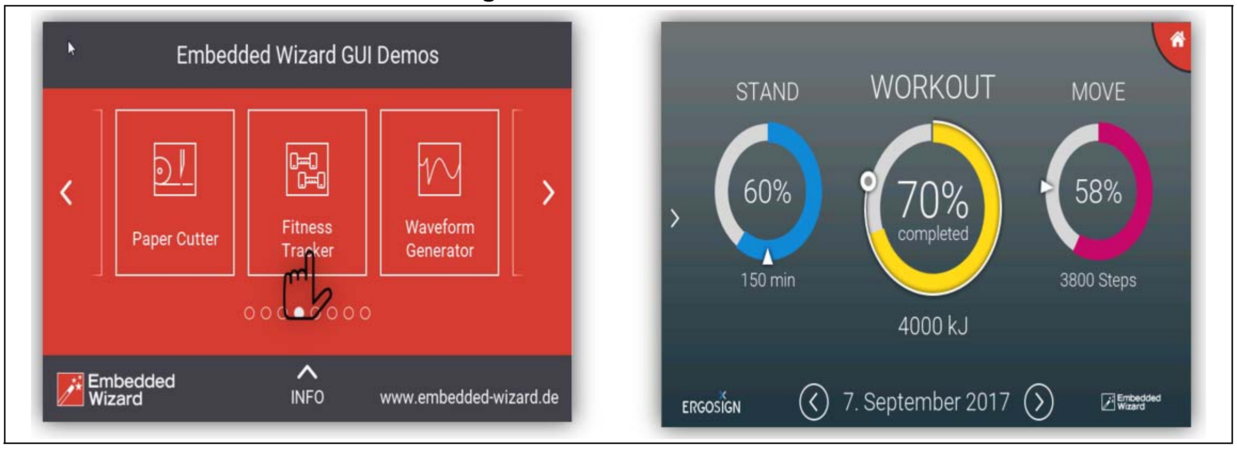


8.3.11 Fitness tracker

Overview

Graphic module showing a fitness tracker application.

Figure 65. Fitness tracker

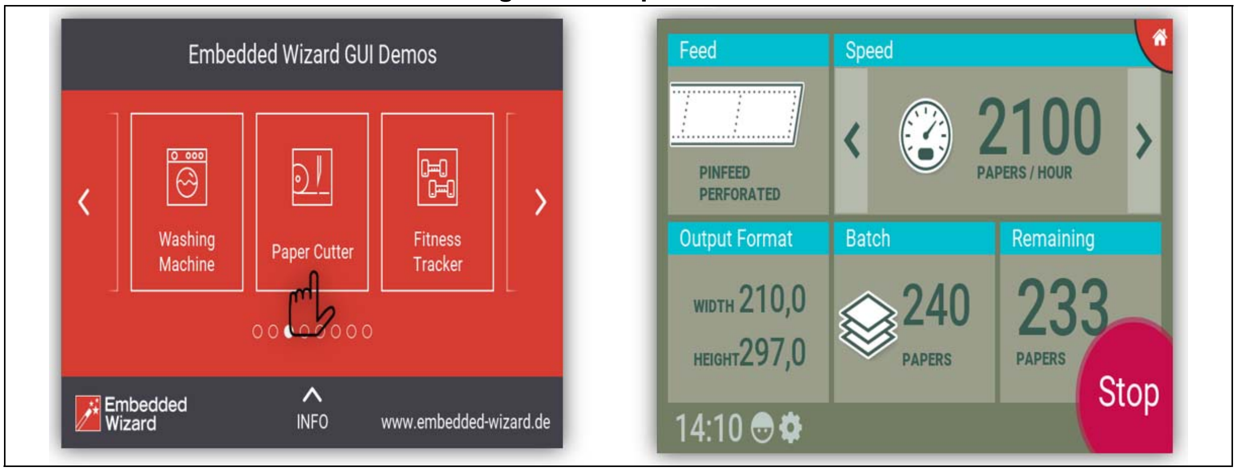


8.3.12 Paper cutter

Overview

Graphic module showing a paper cutter application.

Figure 66. Paper cutter

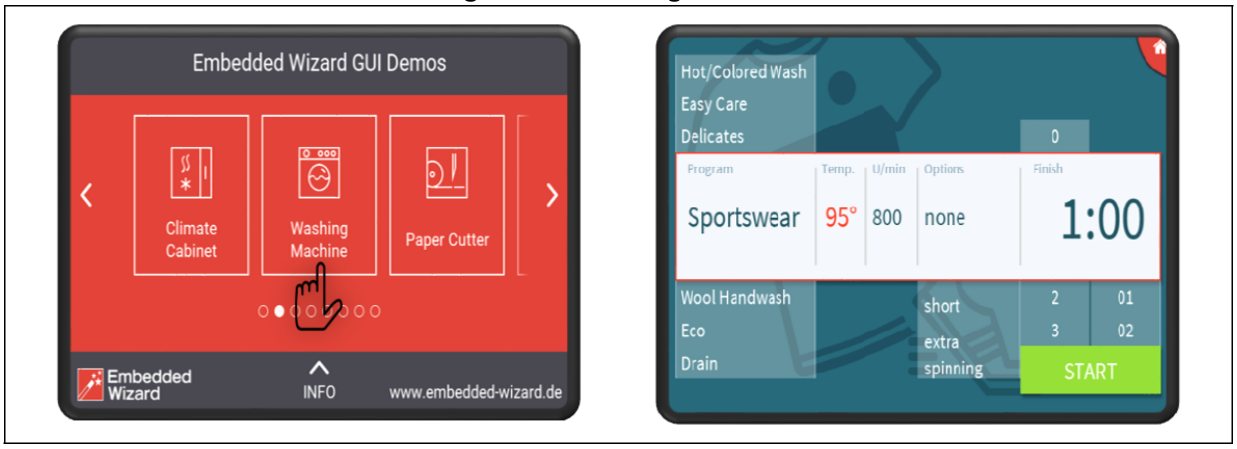


8.3.13 Washing machine

Overview

Graphic module showing Washing machine application.

Figure 67. Washing machine



9 Functional description of the STM32H745I-DISCO demonstration modules

The STM32H745I dual-core Discovery board demonstration aims to highlight the dual-core aspects and the analog features of the STM32H745I dual-core device.

The Demonstration is based on the main menu to switch between:

- Oscilloscope and signals generator application
- EEMBC® CoreMark® application (provided in binary format)
- System information display

9.1 Main graphical interface

Overview

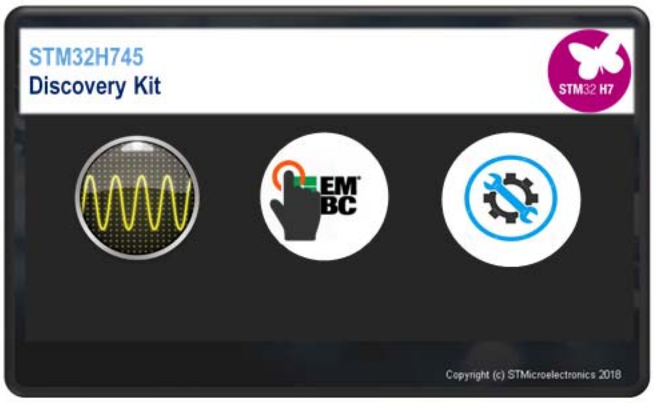
The main graphical interface of the demonstration allows the user to select the application to launch. Here it is possible to run the Oscilloscope, the EEMBC® CoreMark® applications or display the system information.

Content

The graphical interface contains three buttons to select by the user:

- Oscilloscope and signals generator
- EEMBC® CoreMark® benchmark

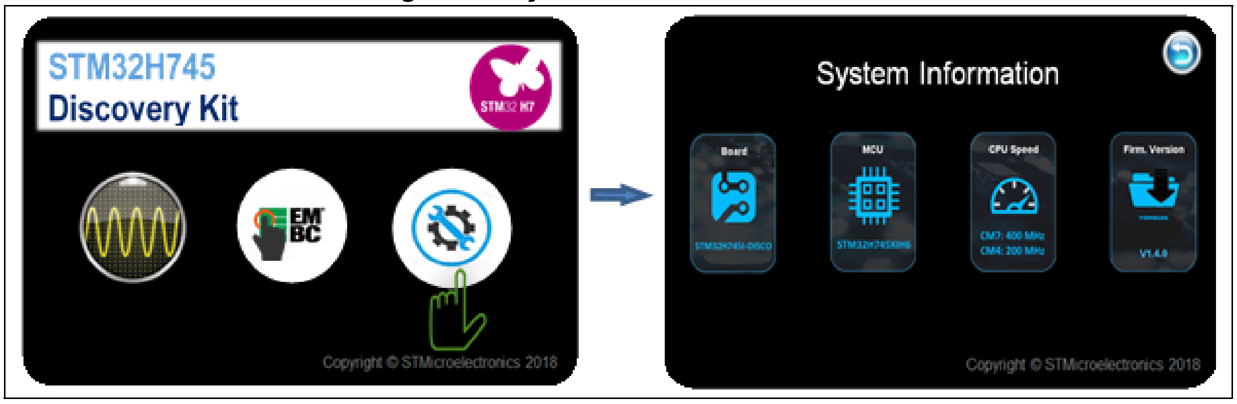
Figure 68. Main menu



9.2 System information

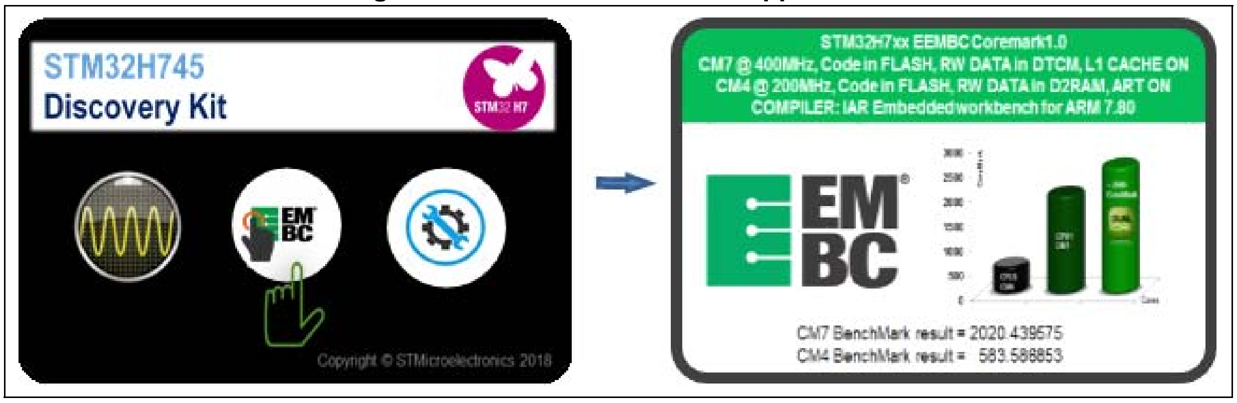
The system information shows the main demonstration information such as the board name, the used device part number, the CPU speeds, and the demonstration firmware version.

Figure 69. System information module



9.3 EEMBC® CoreMark®

Figure 70. EEMBC® CoreMark® application


STM32H745
Discovery Kit
STM32 H7
Copyright © STMicroelectronics 2018
STM32H7xx EEMBC Coremark1.0
CM7 @ 400MHz, Code in FLASH, RW DATA in DTCM, L1 CACHE ON
CM4 @ 200MHz, Code in FLASH, RW DATA in D2RAM, ART ON
COMPILER: IAR Embeddedworkbench for ARM 7.80
EM
BC

```
CM7 BenchMark result = 2020.439575
CM4 BenchMark result =  583.588853
```

EEMBC® CoreMark® is a benchmark that measures the performance of the microcontrollers (MCUs) and central processing units (CPUs) used in embedded systems.

This application shows the STM32H745 device Cortex-M7 and Cortex-M4 scores based on this benchmark.

A simple touch on the screen starts the benchmark simultaneously on both Cortex-M7 and Cortex-M4 and displays the score on real STM32H745 device giving:

- ~2020 CoreMarks for Cortex-M7
- ~583 CoreMarks for Cortex-M4

Algorithm Conditions:

- CPU frequency: Cortex-M7 @ 400MHZ, Cortex-M4 @ 200MHZ
- Code: Cortex-M7 Flash Bank1, Cortex-M4 Flash Bank2
- RW data: Cortex-M7 in DTCM, Cortex-M4 in D2 domain SRAM
- Cortex-M7 L1-Cache ON, Cortex-M4 ART ON
- Number of iterations: Cortex-M7 25000, Cortex-M4 10000
- Compiler: IAR embedded workbench for ARM 7.80

By Default, the STM32H745I-DISCO board is configured to operate in SMPS power mode to reach 400 MHz clock frequency for the Cortex-M7 and 200 MHz for the Cortex-M4.

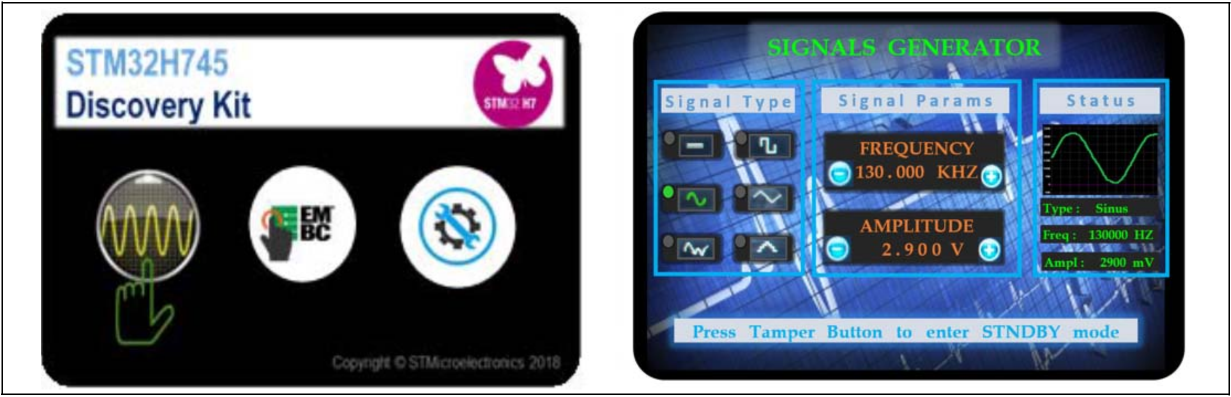
When the board is configured in LDO power mode (with the appropriate solder bridges), the STM32H745I can be overclocked to 480 MHz for the Cortex-M7 and 240 MHz for the Cortex-M4, giving the following CoreMarks performances:

- ~2424 CoreMarks for Cortex-M7
- ~800 CoreMarks for Cortex-M4

Hardware and software steps to switch from SMPS power configuration to LDO are detailed in the file *Demonstration/STM32H745-Discovery_Demo/CM7/Inc/main_common.h* within the STM32CubeH7 MCU package.

9.4 Oscilloscope and signals generator

Figure 71. Oscilloscope and signals generator application



9.4.1 Overview

This application is intended to demonstrate dual-core and analog feature capabilities of the STM32H745 device. Each core is configured to handle a specific analog application:

Cortex-M7 core:

- Execution from the internal Flash memory
- Handle of the signals generator application (SG)

Cortex-M4 core:

- Execution from D2 domain local RAM. Cortex-M4 code is stored in a dedicated Flash section then loaded in the target execution D2 RAM by the Cortex-M7
- Handle of the digital signal oscilloscope (DSO).

The digital signal oscilloscope (DSO) can be used together with the signal generator application (SG) or with an external signal generator. In this case, the signal generator application (SG) can be shut down by pressing the user button in order to put the Cortex-M7 power domain (D1 domain) in STANDBY mode.

Consequently, only the Cortex-M4 domain (D2 domain) keeps running the oscilloscope application. The execution scheme is performed to allow this optimized power configuration:

The Cortex-M4 is executed from D2 RAM. It uses only D2 domain peripherals, to put the D1 domain in standby mode safely.

9.4.2 Features

Analog features:

ADC oversampling, 16 bits resolution, Fast channels mode, dual ADC mode.

Data acquisition and sampling using the STM32H7 peripherals

Dedicated PC GUI:

The Digital Signal Oscilloscope (DSO) comes with a dedicated PC Graphical User Interface. This GUI is available under *PC_Software* within the demonstration Package, to change the oscilloscope settings, visualize the acquired signals and perform mathematical processing.

Communication between the GUI and the STM32H7 is ensured by UART through ST-Link Virtual COM port.

Signal generator:

The signal generator is based on the STM32H745 DAC to generate different signal patterns.

The signal can be then injected into the Oscilloscope using an external wire connection from the DAC output pin to the ADC input pin.

The application provides an HMI on the board LCD display to change the following configurations:

- Signal type: DC, square, sine, triangle, escalator or noise
- Signal frequency: 5,000 kHz to 130,000 kHz
- Signal amplitude: 0.100 V to 3.300 V
- Brief status current output signal

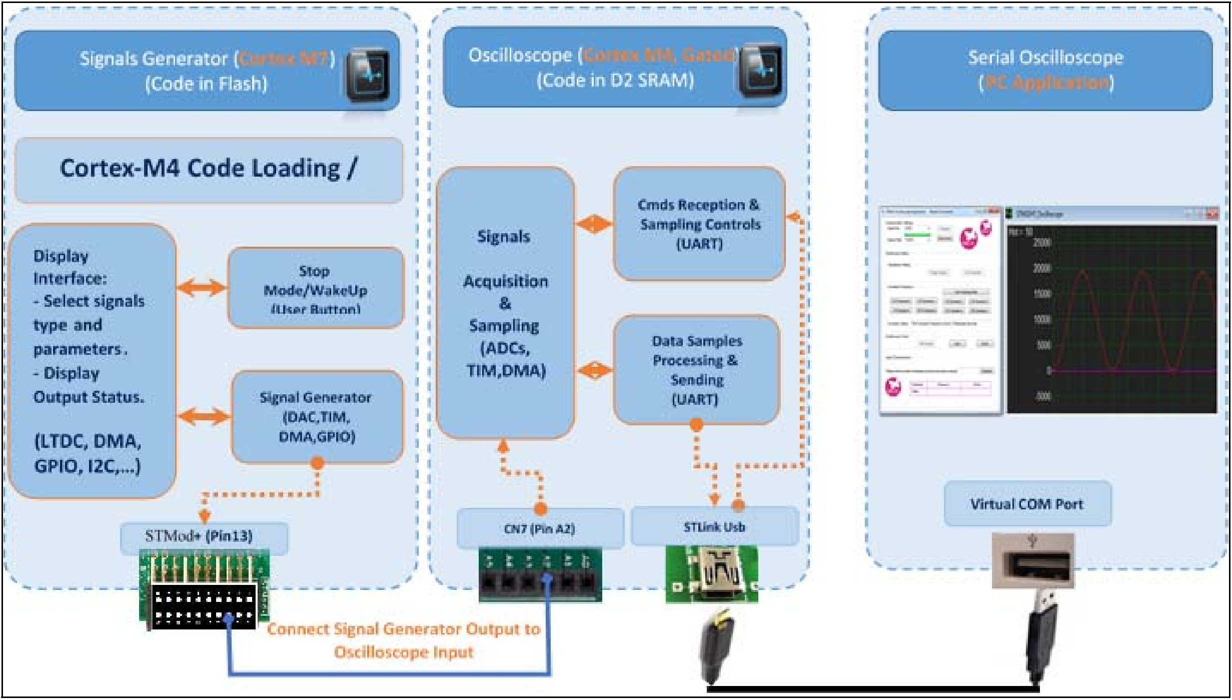
Dual-core aspect:

The demonstration is split between the STM32H745 cores (Cortex-M4 and Cortex-M7) as follow:

- Cortex-M4 to handle data acquisition and communication with the PC GUI
- Cortex-M7 to handle the auto-test mode (LCD HMI, DAC control)
  - Cortex-M7 goes to low power mode when the auto-test mode is switched off.
  - User button let the STM32H745 D1 power domain enter or exit standby mode
  - The entire STM32H745 D1 power domain is Standby in stop mode including the Flash memory, Cortex-M4 continues running his code from D2 domain RAM.

The figure below summarizes the whole process and shows how the different demonstration modules are split between the Cortex-M7 and Cortex-M4.

Figure 72. Oscilloscope and signals generator overview


Signals Generator (Cortex M7)
(Code in Flash)
Cortex-M4 Code Loading /
Display Interface:
- Select signals type and parameters.
- Display Output Status.
(LTDC, DMA, GPIO, I2C,...)
Stop Mode/WakeUp (User Button)
Signal Generator (DAC,TIM, DMA,GPIO)
STMod+ (Pin13)
Oscilloscope (Cortex M4, Gated)
(Code in D2 SRAM)
Signals Acquisition & Sampling (ADCs, TIM,DMA)
Cmds Reception & Sampling Controls (UART)
Data Samples Processing & Sending (UART)
CN7 (Pin A2)
STLink Usb
Serial Oscilloscope
(PC Application)
Virtual COM Port
Connect Signal Generator Output to Oscilloscope Input

9.4.3 Hardware and software setup

1. Install the PC GUI application on the PC

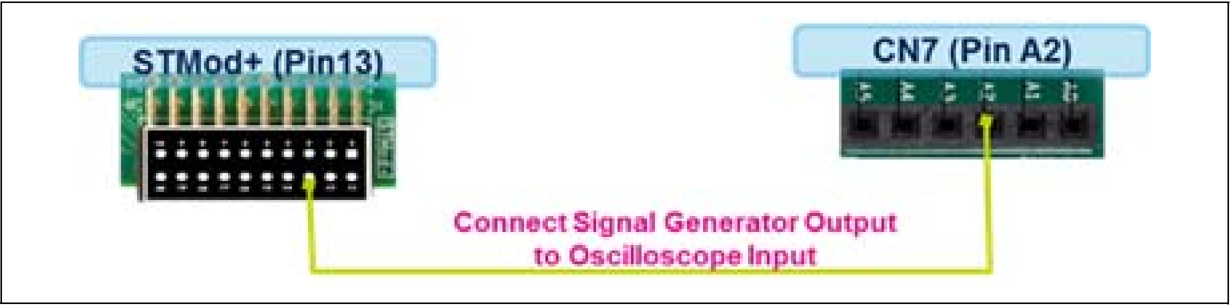
- This application is used to:
  a) Configure the ADC parameters
  b) Receive and visualize samples acquired by the ADC and sent by the Cortex-M4
- The communication between the PC application and the STM32H745I-DISCO is ensured by the ST-Link USB connector used as Virtual COM port
- Connect the STM32H745I-DISCO USB ST-LINK port to the PC

2. STM32H745I-DISCO HW setup

In order to inject the signal generator output to the oscilloscope input, use an external wire connection from the on-board STMod+ (pin13) to CN7 (pin A2) as shown in *Figure 73*:

- STMod+ (pin13): the signal generator (DAC) output pin
- CN7 (pin A2): this is the Oscilloscope (ADC) input pin

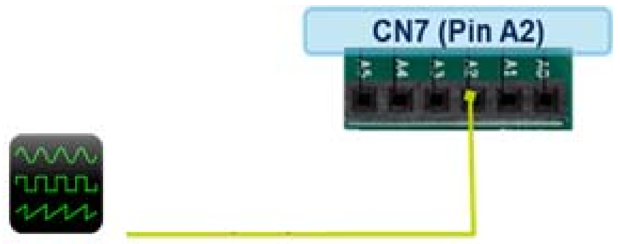
Figure 73. Signal generator to oscilloscope connection



The user may choose to inject his own signal to the oscilloscope, in this case:

- Remove the STMod+ (pin13) to CN7 (pin A2) connection
- Inject the signal to be sampled by the oscilloscope: connect an external signal source to CN7 (pin A2)
- The signal parameter must be as follow:
  - Amplitude: from 0 to 3.3 V
  - Frequency: up to 1 MHz

Figure 74. External signal to oscilloscope connection



9.4.4 How to use

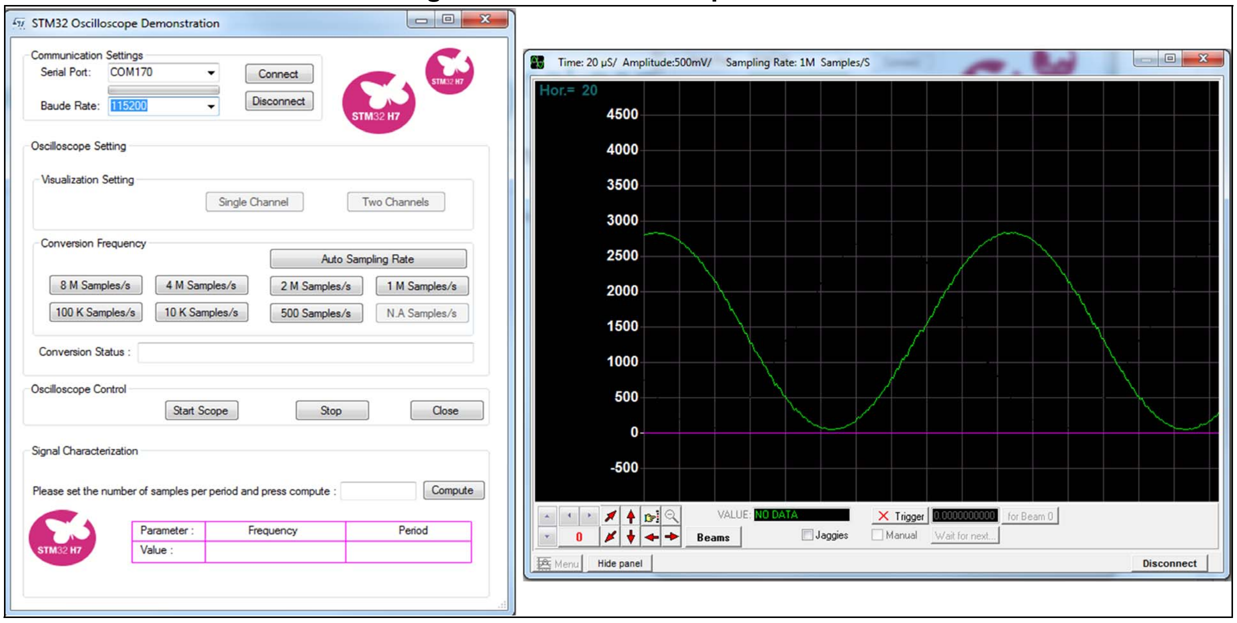
1. PC Software Side:

Start the PC oscilloscope application

- In the *Communication Settings*
  - Select the *Serial Port* corresponding to the ST-Link Virtual COM
  - Select the *Baud Rate* 115200
  - Click on *Connect*

Click on `Start Scope` to visualize the input signal.

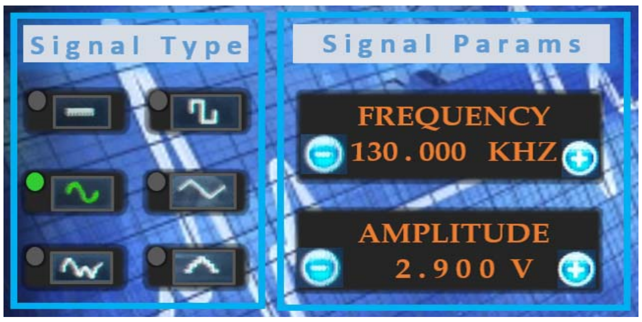
Figure 75. PC oscilloscope screen



2. STM32H745I-DISCO board side:

The user may select the type, amplitude, and frequency of the signal generator from the LCD by touching the corresponding buttons.

Figure 76. STM32H745I-DISCO board screen



If there is no user activity on the touchscreen for 15 seconds, the signal generator goes to auto mode, randomly changing the signal type, frequency, and amplitude.

The user may switch off the Cortex-M7 (D1 domain goes to standby mode):

- Press the user button on the STM32H745I-DISCO board to put D1-Domain in standby mode
  - All the STM32H745 D1 power domain goes to standby (including the Cortex-M7, the internal Flash memory, and the LCD controller).
  - The D2 domain remains in run mode with Cortex-M4 running from the local D2 RAM.
  - The signal generator output is switched off (DAC is stopped) and the LCD is shutdown.
  - The Cortex-M4 still executes the oscilloscope application.
  - An external signal must be connected to the oscilloscope (ADC) input (pin A2 on CN7 connector).
- Pressing again the user button wakes up the Cortex-M7 D1 power Domain and resumes the signal generator application

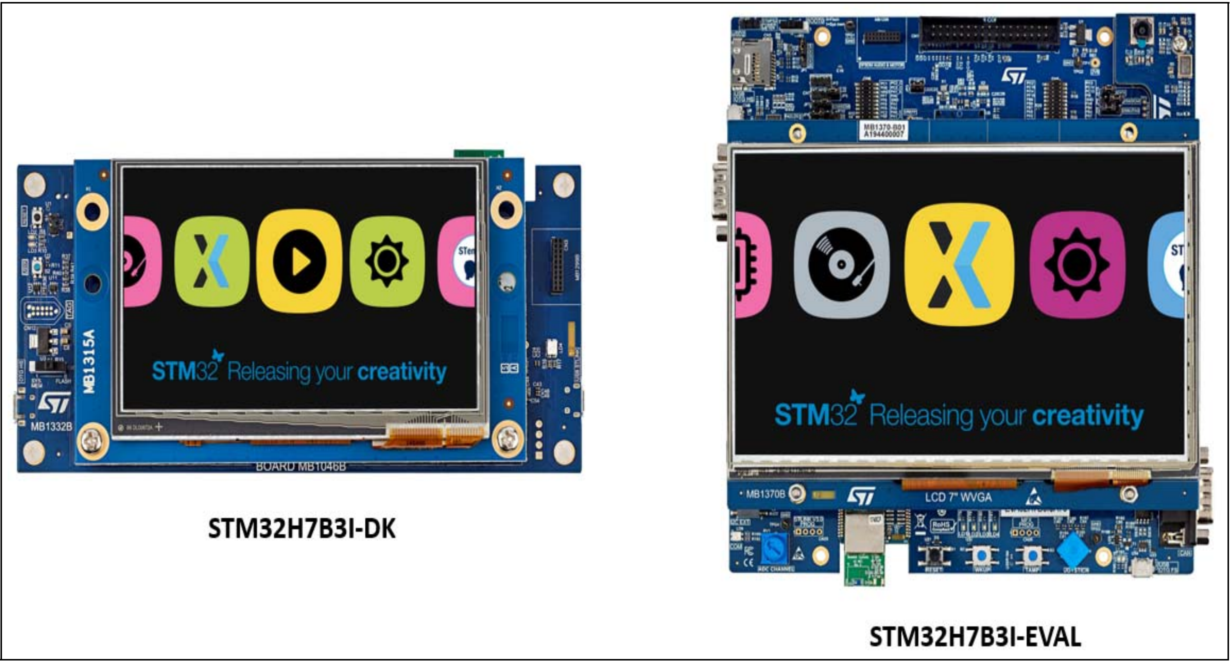
10 Functional description of the STM32H7B3I-DK and STM32H7B3I-EVAL demonstration modules

The Demonstration is based on the main Menu Launcher developed with TouchGFX API and features the following modules:

- Audio Player
- Video Player (Only STM32H7B3I-DK)
- Clock & Weather
- TouchGFX demo
- System information
- Embedded Wizard
- STemWin

Figure 77 illustrates the Menu Launcher content on STM32H7B3I-DK and STM32H7B3I-EVAL boards.

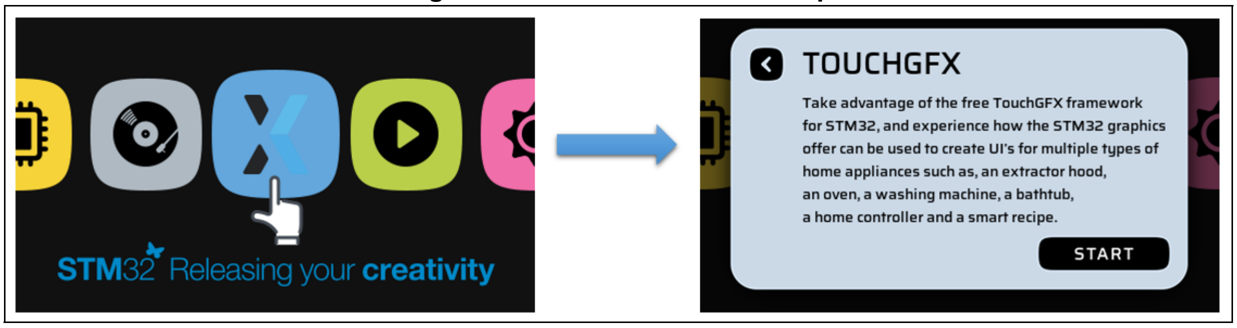
Figure 77. Menu Launcher content


STM32 Releasing your creativity
STM32H7B3I-DK
STM32 Releasing your creativity
LCD 7" WVGA
STM32H7B3I-EVAL

Functional description

1. To run any demo, select the associated icon or scroll the icons bar until the desired demo icon is in the middle then click on the selected demo icon.
2. A demo overview is displayed.
3. Click on Start to run a demo or on exit to return to Menu Launcher.

Figure 78. Menu Launcher startup



Take advantage of the free TouchGFX framework for STM32, and experience how the STM32 graphics offer can be used to create UI's for multiple types of home appliances such as, an extractor hood, an oven, a washing machine, a bathtub, a home controller and a smart recipe.

10.1 Audio Player module

10.1.1 Overview

The audio player module is developed with TouchGFX API and provides a complete audio solution based on the STM32H7 Series and delivers a high-quality music experience. It supports playing music in WAV and MP3 audio formats.

10.1.2 Features

- Audio format: WAV and MP3 formats
- Progress bar navigation support
- Repeat and shuffle music support
- Volume control support
- Supports low power mode: the system can switch to low power mode while keeping the music playing for more power saving.
- Auto-repeat enabled when application switch to low power mode

10.1.3 Notes

- The user can exit from low power mode by pressing the wake-up button on the board (User button for STM32H7B3I-DK and wake-up button for STM32H7B3I-EVAL).
- Audio files need to be stored in the SD card
- The application returns to Menu Launcher if SD card is unplugged after the demo is started.
- For the STM32H7B3I-EVAL board, make sure that these jumpers JP6, JP7, JP8, JP9, JP11, and JP18 are in I2S6 position (JP16 and JP39 are removed) in order to keep the audio working correctly.

Figure 79. Audio player module

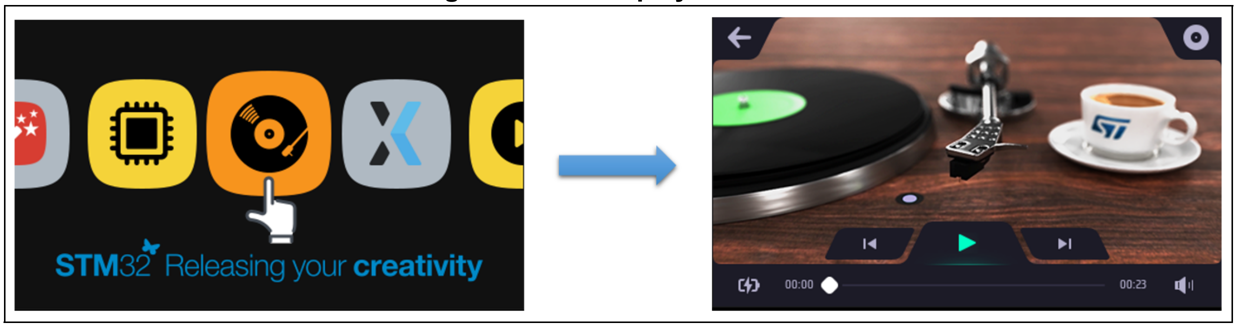


Figure 80. Select music from a playlist

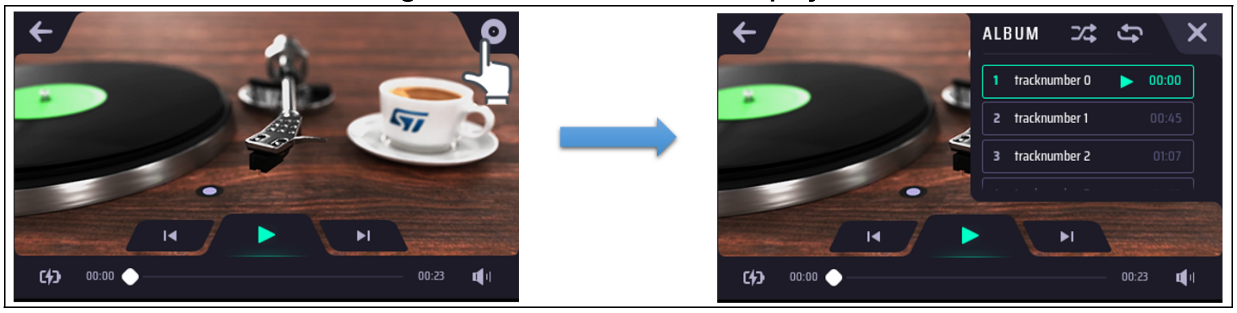
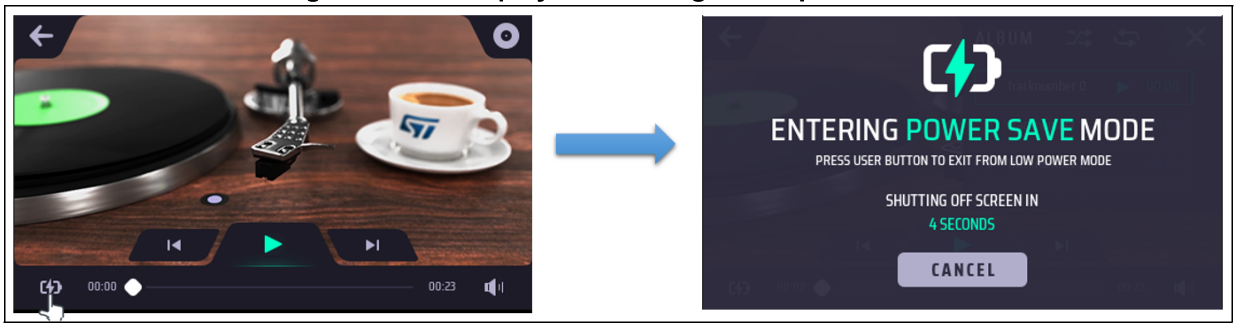


Figure 81. Audio player switching to low power mode



ENTERING POWER SAVE MODE

PRESS USER BUTTON TO EXIT FROM LOW POWER MODE

SHUTTING OFF SCREEN IN
4 SECONDS

10.2 Video Player module (Only STM32H7B3I-DK)

10.2.1 Overview

The video player module provides a video solution based on the STM32H7 Series and TouchGFX API. It supports playing a movie in AVI format.

10.2.2 Features

- Video format: AVI mjpeg or wav format (audio and video)
- Displays all supported videos available on media storage with preview
- Volume control support
- Displays total and elapsed times
- Info bar and video controls are auto-hidden after few seconds of no interaction

10.2.3 Notes

- The Video files used with this module need to be stored in the SD card.
- Only 480x272 video resolution is supported.
- Progress bar navigation is not supported
- The application returns to Menu Launcher if SD card is unplugged after the demo is started.

Figure 82. Video player module

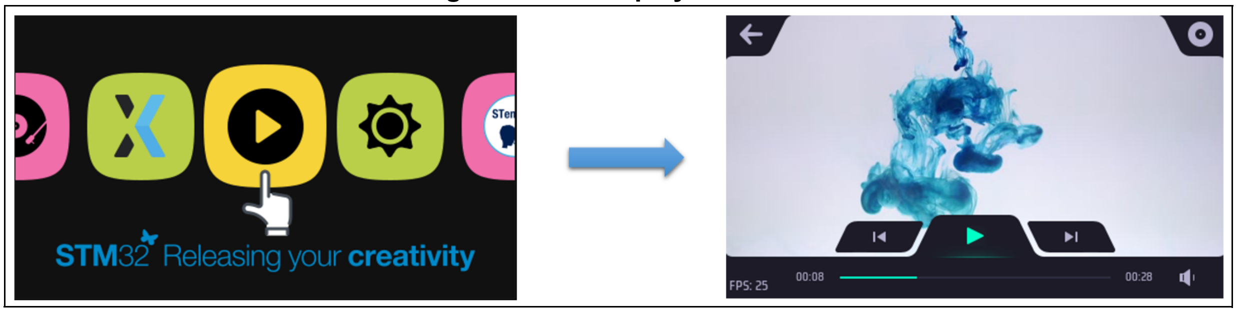
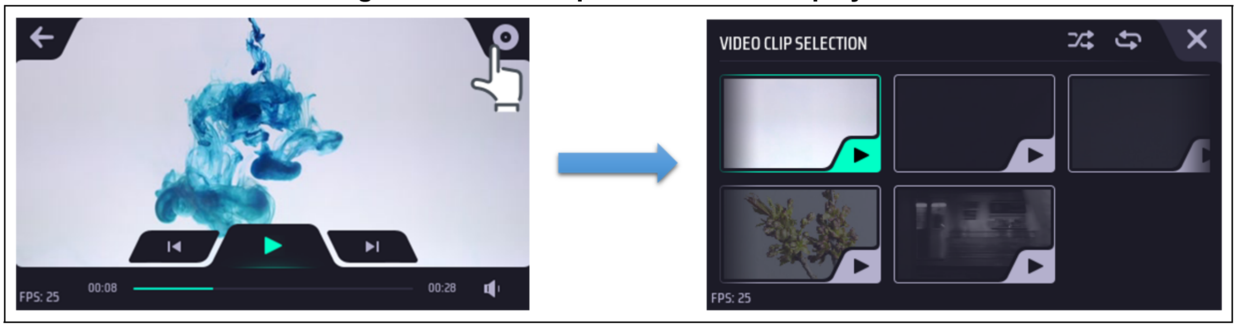


Figure 83. Video clip selection from a playlist



10.3 Clock & Weather module

10.3.1 Overview

The clock and weather module allows displaying the time, date and forecast for different cities using the WIFI module.

When running the application, a list of available WIFI access points is displayed in the settings menu. The user can choose an access point and connect to it by typing the password using the keyboard.

When the connection is established, the date, time and forecast for each city is retrieved from the server.

The user can add custom cities (maximum 3 cities) using microSD™. The user must create a configuration file named *cw_cfg.ini* in the microSD™ card which contains the name of the chosen city (location = city_name).

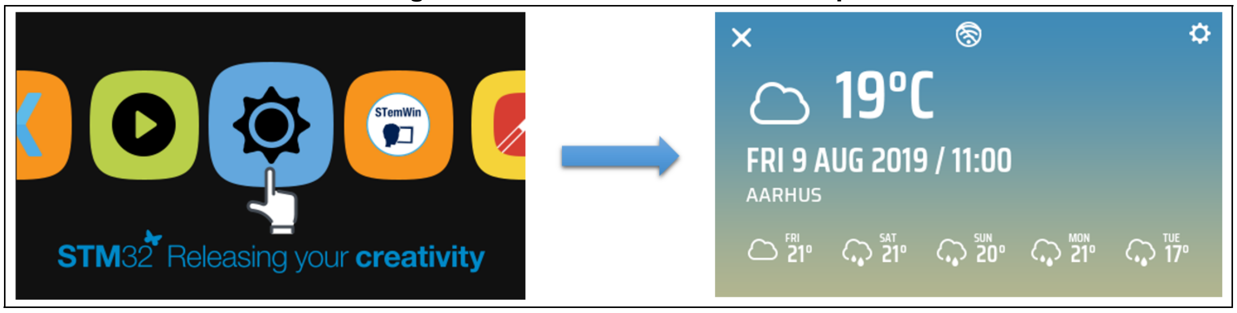
10.3.2 Notes

- For STM32H7B3I-EVAL board:
  - Clock&Weather demonstration supports only one city (Paris by default) and the user can change it by adding his custom city in the microSD™ card.
  - Additional information such as wind speed, wind direction, pressure, humidity and rain volume were added in the STM32H7B3I-EVAL board for better user experience.

10.3.3 Functional description

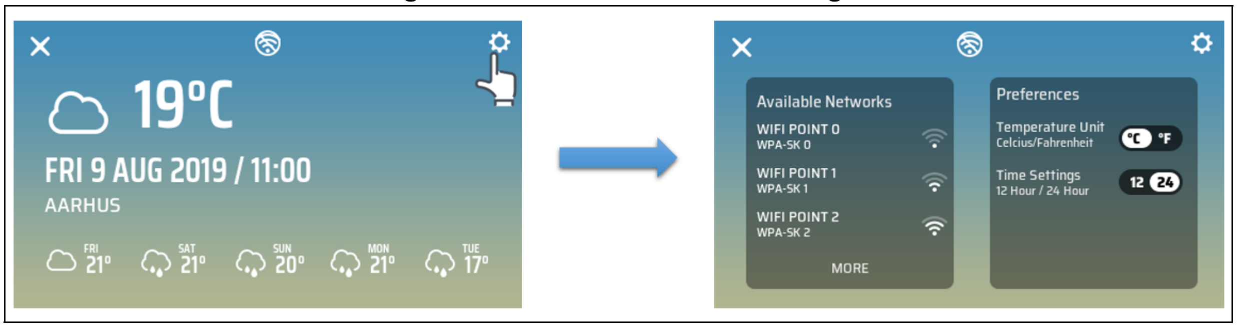
1. Start the module by touching the clock and weather icon.

Figure 84. Clock and weather startup



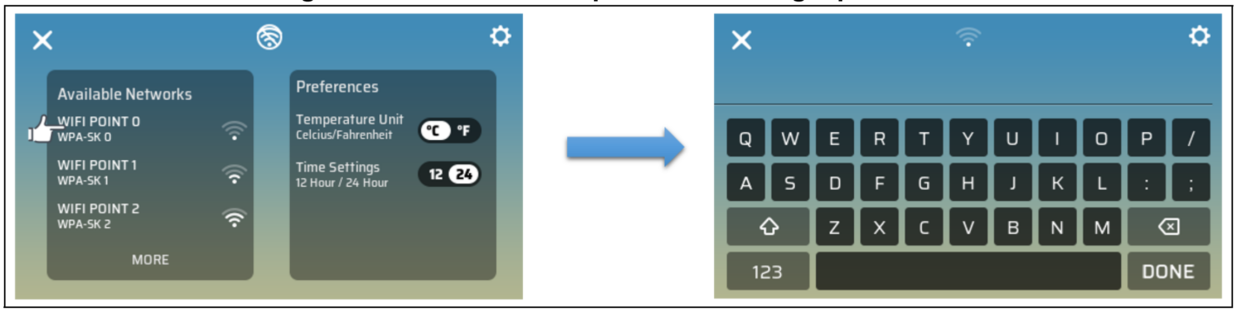
2. Press the Settings button to display available access points, temperature settings, and clock format.

Figure 85. Clock and weather settings



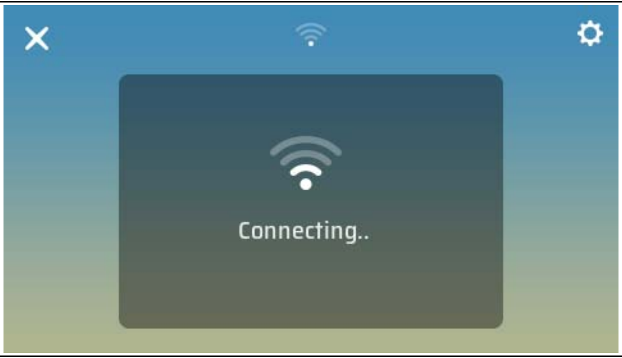
3. Choose a valid access point and connect to it by entering the password.

Figure 86. Select access point and setting a password



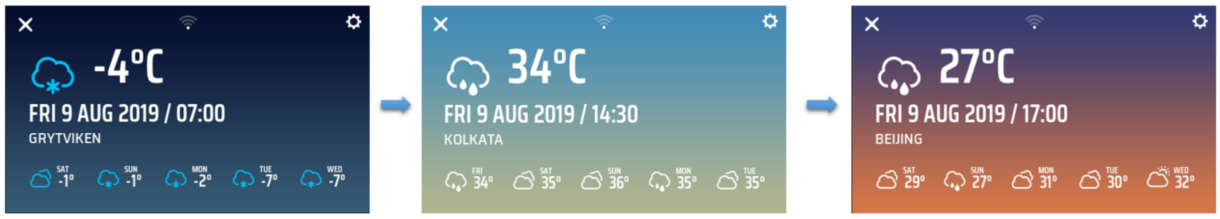
4. Connecting to an access point and collecting data from the server.

Figure 87. Connecting to access point and collecting data



5. After connecting and retrieving data successfully, slide to right or left to display time, date and forecast for each city.

Figure 88. Time, date and forecast for different cities


-4°C
FRI 9 AUG 2019 / 07:00
GRYTVIKEN
SAT -1° SUN -1° MON -2° TUE -7° WED -7°
34°C
FRI 9 AUG 2019 / 14:30
KOLKATA
FRI 34° SAT 35° SUN 36° MON 35° TUE 35°
27°C
FRI 9 AUG 2019 / 17:00
BEIJING
SAT 29° SUN 27° MON 31° TUE 30° WED 32°

10.3.4 Use private key for weather API calls

Overview

Each weather API call needs an API key to get forecast data from the forecast weather service provider (openweathermap.org).

API call:

http://api.openweathermap.org/data/2.5/forecast?id=524901&APPID={APIKEY}

Parameters:

APPID {APIKEY} is the user’s unique API key

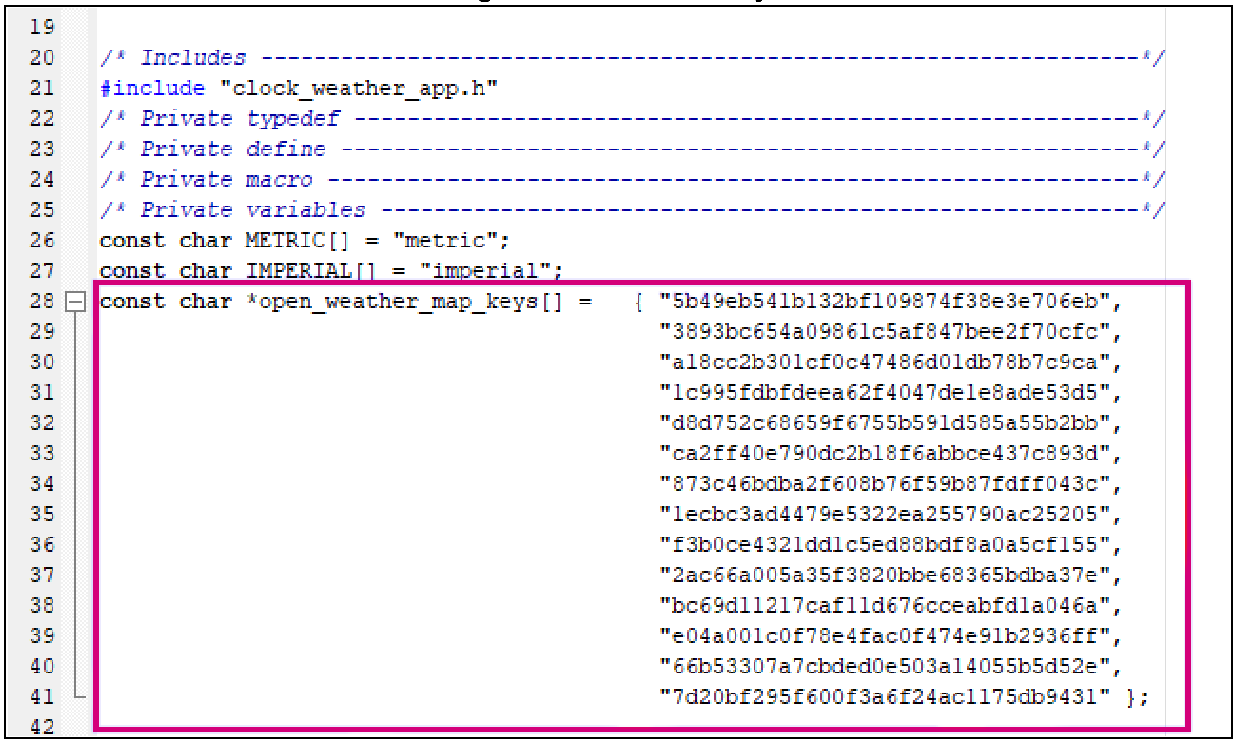
Example of API call:

api.openweathermap.org/data/2.5/forecast?id=524901&APPID=1111111111

Steps to change the API key in code source application:

1. In order to get his private key, the user needs to sign up on openweathermap.org site.
2. In clock_weather_app.c file, he must find the open_weather_map_keys[] tab and add his custom key there.

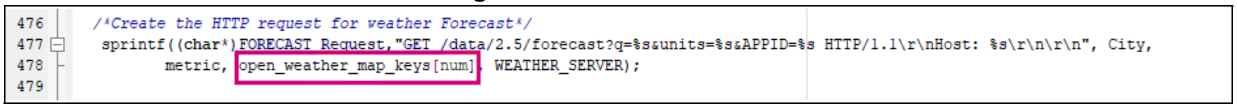
Figure 89. List of API keys



```
/* Includes ------------------------------------------------------------------*/
#include "clock_weather_app.h"
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
const char METRIC[] = "metric";
const char IMPERIAL[] = "imperial";
const char *open_weather_map_keys[] =   { "5b49eb541b132bf109874f38e3e706eb",
                                          "3893bc654a09861c5af847bee2f70cfc",
                                          "a18cc2b301cf0c47486d01db78b7c9ca",
                                          "1c995fdbfdeea62f4047de1e8ade53d5",
                                          "d8d752c68659f6755b591d585a55b2bb",
                                          "ca2ff40e790dc2b18f6abbce437c893d",
                                          "873c46bdba2f608b76f59b87fdff043c",
                                          "1ecbc3ad4479e5322ea255790ac25205",
                                          "f3b0ce4321dd1c5ed88bdf8a0a5cf155",
                                          "2ac66a005a35f3820bbe68365bdba37e",
                                          "bc69d11217caf11d676cceabfd1a046a",
                                          "e04a001c0f78e4fac0f474e91b2936ff",
                                          "66b53307a7cbded0e503a14055b5d52e",
                                          "7d20bf295f600f3a6f24ac1175db9431" };
```

3. Use the custom key in any weather API call.

Figure 90. Weather API call



```
/*Create the HTTP request for weather Forecast*/
 sprintf((char*)FORECAST_Request,"GET /data/2.5/forecast?q=%s&units=%s&APPID=%s HTTP/1.1\r\nHost: %s\r\n\r\n", City,
         metric, open_weather_map_keys[num], WEATHER_SERVER);
```

10.4 TouchGFX demo

10.4.1 STM32H7B3I-DK board

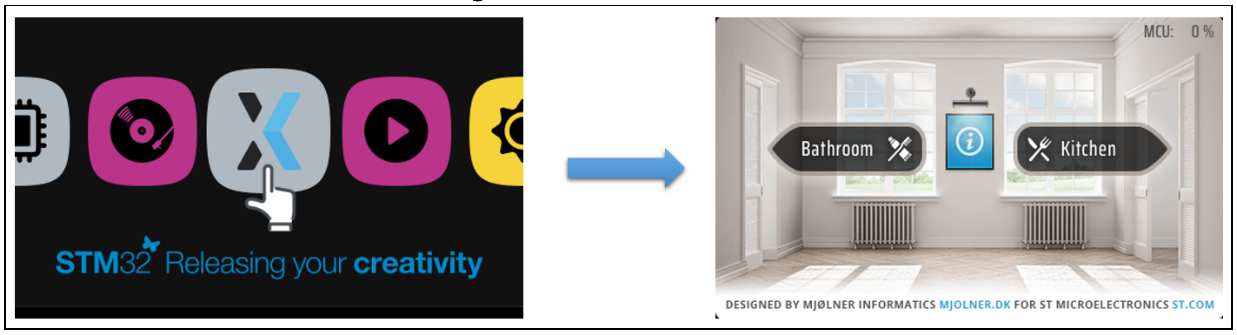
Overview

TouchGFX demo consists of a home control module, which allows the user to control different connected devices for different rooms in the house.

Functional description

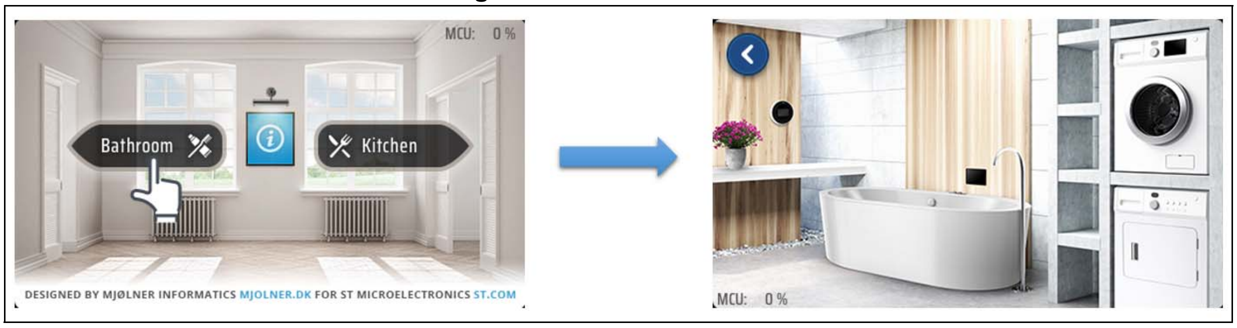
1. Start the module by touching the TouchGFX demo icon.

Figure 91. TouchGFX demo


STM32 Releasing your creativity
MCU: 0 %
Bathroom
Kitchen
DESIGNED BY MJØLNER INFORMATICS MJOLNER.DK FOR ST MICROELECTRONICS ST.COM

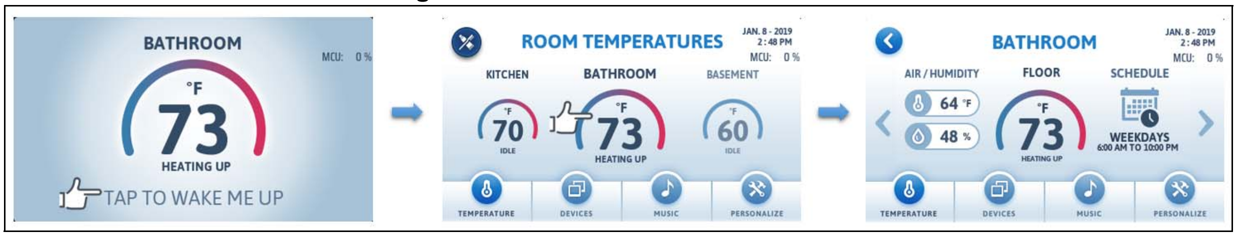
2. Select the bathroom button to control connected devices in the bathroom.

Figure 92. Bathroom menu


MCU: 0 %
Bathroom
Kitchen
DESIGNED BY MJØLNER INFORMATICS MJOLNER.DK FOR ST MICROELECTRONICS ST.COM
MCU: 0 %

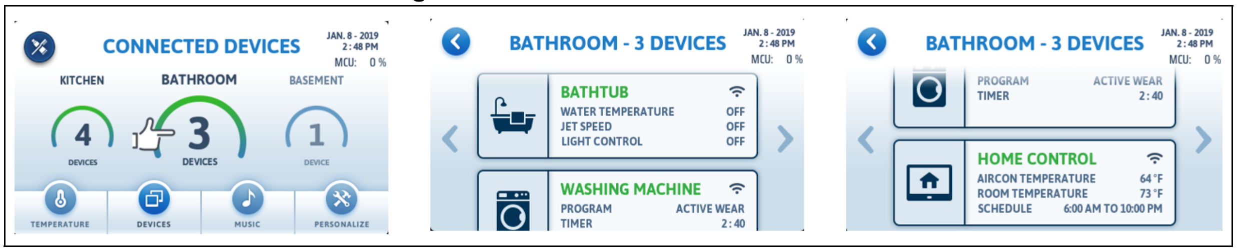
3. Select the thermostat to control the temperature.

Figure 93. Thermostat control menu



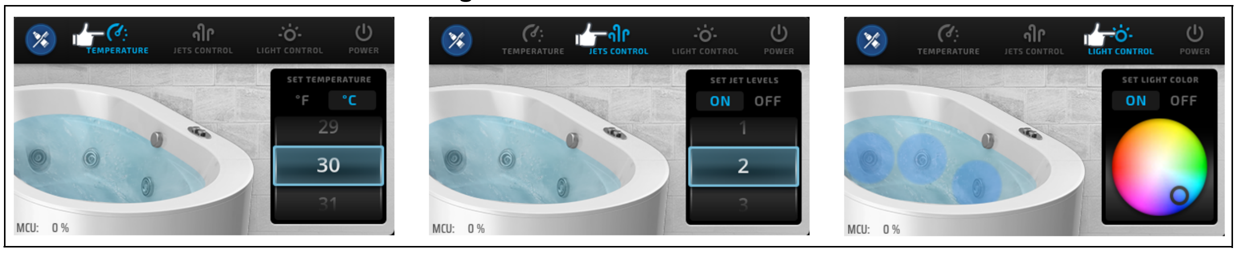
4. The user can examine the connected devices status by selecting the device menu.

Figure 94. Connected devices status



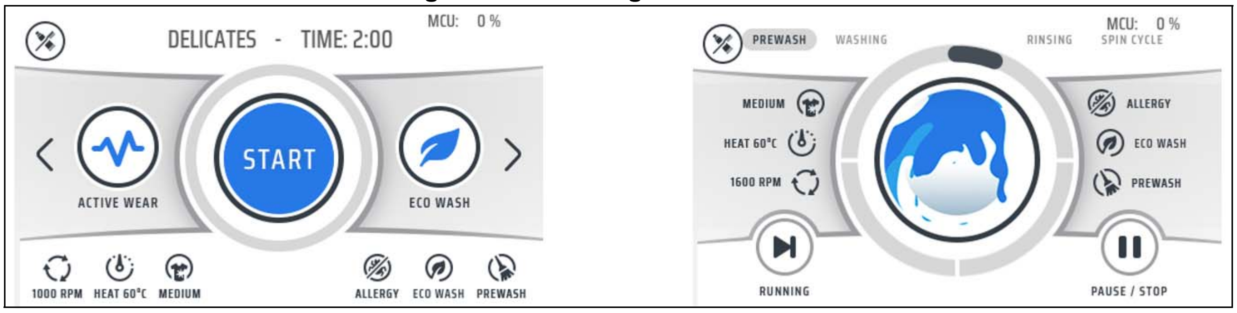
5. In the bathroom menu, select the bath. A control panel shows up to configure water temperature, jets level, and light color.

Figure 95. Bath control menu



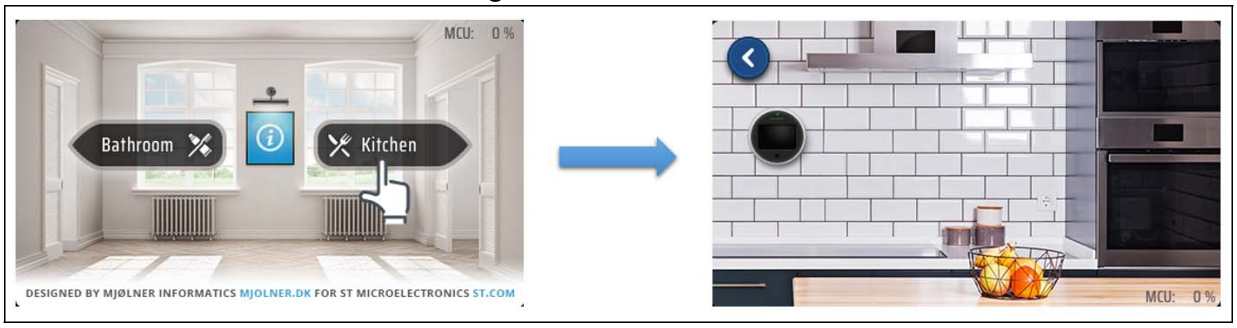
6. Select the washing machine to control and monitor washing cycles.

Figure 96. Washing machine menu



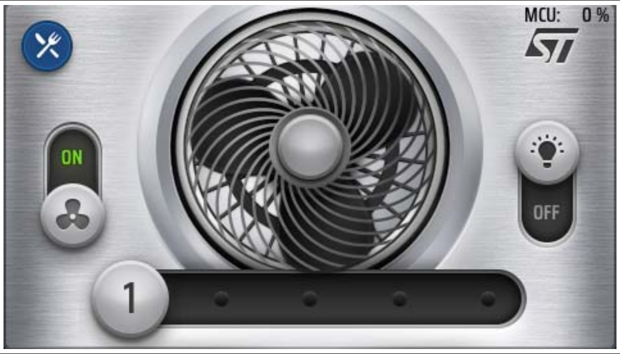
7. Select the kitchen button in the TouchGFX demo menu to get access to connected devices in the kitchen.

Figure 97. Kitchen menu


MCU: 0 %
Bathroom
Kitchen
DESIGNED BY MJØLNER INFORMATICS MJOLNER.DK FOR ST MICROELECTRONICS ST.COM
MCU: 0 %

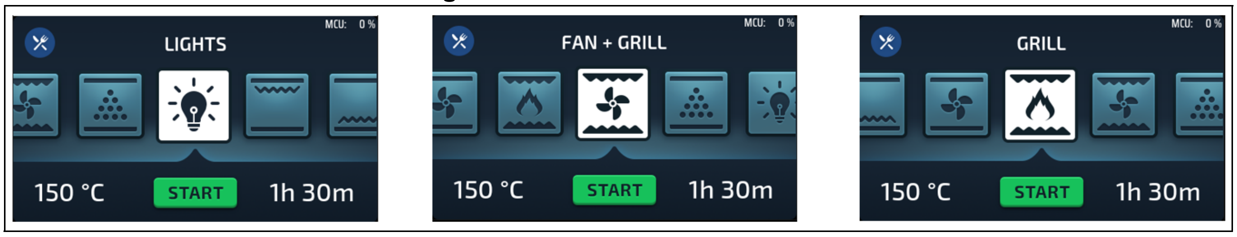
8. Select the range hood to control the lights and hood blower speed.

Figure 98. Range hood control



9. Select the cooker to control the heat, lights, fan, and grill.

Figure 99. Cooker control menu



10.4.2 STM32H7B3I-EVAL board

Overview

The TouchGFX demo in STM32H7B3I-EVAL board features the following modules:

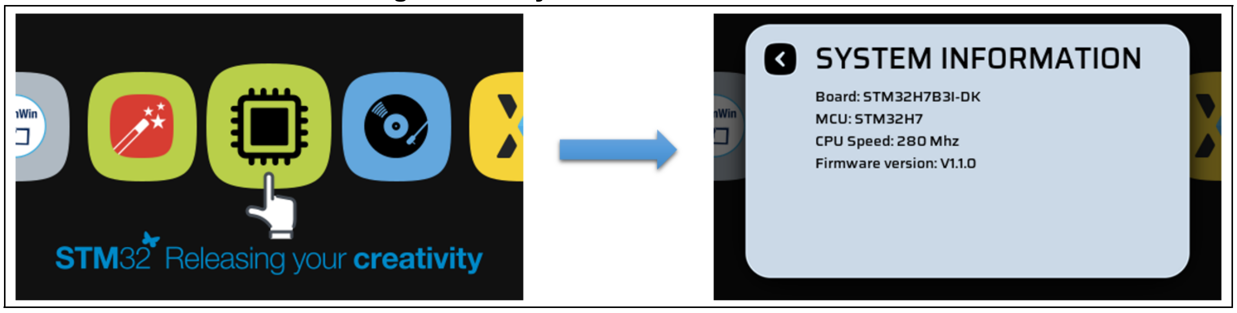
- Graphic effect
- Time and calendar
- 2048 puzzle game
- Home control
- External hardware
- Bird Eat Coin game

For further details about these modules please refer to *Section 8.2: TouchGFX demonstration*.

10.5 System information

The system information shows the main demonstration information such as the board name, the used device part number, the CPU speeds, and the demonstration firmware version.

Figure 100. System information module



Board: STM32H7B3I-DK
MCU: STM32H7
CPU Speed: 280 Mhz
Firmware version: V1.1.0

10.6 STemWin

10.6.1 Overview

The STemWin demo allows the user to switch between these modules:

- Rocket game
- Graphic effect

For more details about these modules, please refer to *Section 8.1.5: Graphic effect* and *Section 8.1.3: Rocket game*.

10.7 Embedded Wizard

10.7.1 Overview

The embedded wizard demonstration is available in binary format.

The embedded wizard demonstration allows the user to switch between these modules:

- Graphic effect
- Graphics accelerator
- Waveform generator
- Charts demonstration
- Climate cabinet
- Brick game
- Fitness tracker
- Paper cutter
- Washing machine

For more details about these modules, please refer to section 8.3 (Embedded wizard demonstration).

10.8 Known limitations

10.8.1 Clock & weather module

1. The Forecast weather service provider (OpenWeather map) modifies the data format by introducing new fields. This impacts data collection: city names and temperature values may sometimes be wrong. This issue must be fixed in the next release of the demo.
   ST recommends downloading the new binary available on the ST website for users who want to upgrade the demonstration binary.
2. API keys are used in each weather API call, so sometimes the user may face a problem when collecting forecast data from the server due to the limited key number used by the application.
   When facing this problem, ST recommends trying connecting again in order to get forecast data.
   To use a private key, refer to *Section 10.3.4*, which explains how to integrate a custom key in application source code.
3. For STM32H7B3I-DK board: Auto scan feature is not enabled after connecting to an access point so make sure that the Wi-Fi® device is activated before connecting otherwise restart the demonstration to detect new APIs.

Revision history

Table 12. Document revision history

Date	Revision	Changes
8-Dec-2017	1	Initial version
12-Jun-2019	2	Updated *Introduction*, *Section 8* to add newly supported boards and applications Added *Section 9* specific to STM32H745I-DISCO
12-Mar-2020	3	Updated cover image with STM32H7B3I-DK and STM32H7B3I-EVAL boards Added *Section 10: Functional description of the STM32H7B3I-DK and STM32H7B3I-EVAL demonstration modules*

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to *www.st.com/trademarks*. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved