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, allowing them to be reused separately in standalone applications. All these modules are managed by the STM32CubeH7 demonstration kernel allowing 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 can be used 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 boards.
# Contents

1. **STM32Cube overview** ................................................. 8

2. **Global architecture** .................................................. 9

3. **Kernel description** ................................................... 10
   - 3.1 Overview .......................................................... 10
   - 3.2 Kernel initialization ............................................. 11
   - 3.3 Kernel processes and tasks .................................... 12
   - 3.4 Kernel graphical aspect ........................................ 12
   - 3.5 Kernel menu management ....................................... 13
   - 3.6 Module manager .................................................. 15
   - 3.7 Backup and settings configuration ............................ 16
   - 3.8 Storage units ..................................................... 17
   - 3.9 Adding binary demonstration ................................... 19
   - 3.10 Demonstration repository ..................................... 21
   - 3.11 Kernel components ............................................. 22
   - 3.12 Kernel core files .............................................. 22
   - 3.13 Hardware settings .............................................. 23

4. **How to create a new module** ...................................... 24
   - 4.1 Creating the graphical aspect ................................. 24
   - 4.2 Graphics customization ......................................... 24
   - 4.3 Module implementation ......................................... 25
   - 4.4 Adding a module to the main desktop ........................ 26

5. **Demonstration customization and configuration** ............ 27
   - 5.1 LCD configuration .............................................. 27
   - 5.2 Layer management ............................................. 27
   - 5.3 BSP customization ............................................... 28
     - 5.3.1 SDRAM configuration ........................................ 28
     - 5.3.2 Touchscreen configuration .................................. 29
List of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>File system interface functions</td>
<td>17</td>
</tr>
<tr>
<td>Table 2</td>
<td>Kernel components list</td>
<td>22</td>
</tr>
<tr>
<td>Table 3</td>
<td>Kernel core files list</td>
<td>22</td>
</tr>
<tr>
<td>Table 4</td>
<td>Jumpers for demonstration boards</td>
<td>23</td>
</tr>
<tr>
<td>Table 5</td>
<td>LCD frame buffer locations</td>
<td>28</td>
</tr>
<tr>
<td>Table 6</td>
<td>Modules footprint</td>
<td>35</td>
</tr>
<tr>
<td>Table 7</td>
<td>RAM requirements for some JPEG resolutions</td>
<td>35</td>
</tr>
<tr>
<td>Table 8</td>
<td>MemoSTEMWin components memory requirements</td>
<td>36</td>
</tr>
<tr>
<td>Table 9</td>
<td>Widget memory requirements</td>
<td>36</td>
</tr>
<tr>
<td>Table 10</td>
<td>Audio player module controls</td>
<td>40</td>
</tr>
<tr>
<td>Table 11</td>
<td>Video player module controls</td>
<td>43</td>
</tr>
<tr>
<td>Table 12</td>
<td>Document revision history</td>
<td>56</td>
</tr>
</tbody>
</table>
## List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>STM32Cube block diagram</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2</td>
<td>STM32CubeH7 demonstration architecture</td>
<td>9</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Kernel components and services</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Splash screen</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Demonstrations main menu</td>
<td>13</td>
</tr>
<tr>
<td>Figure 6</td>
<td>STMWin demonstration main menu</td>
<td>14</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Functionalities and properties of modules</td>
<td>15</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Available storage units</td>
<td>17</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Software architecture</td>
<td>18</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Demonstration memory mapping</td>
<td>19</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Demonstration folder structure</td>
<td>21</td>
</tr>
<tr>
<td>Figure 12</td>
<td>STM32H743-EVAL demonstration board</td>
<td>23</td>
</tr>
<tr>
<td>Figure 13</td>
<td>GUIBuilder overview</td>
<td>24</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Graphics customization</td>
<td>25</td>
</tr>
<tr>
<td>Figure 15</td>
<td>LCDConf.c location</td>
<td>27</td>
</tr>
<tr>
<td>Figure 16</td>
<td>SDRAM initialization</td>
<td>28</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Touch screen configuration</td>
<td>29</td>
</tr>
<tr>
<td>Figure 18</td>
<td>STM32H7 Series system architecture</td>
<td>30</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Performance of STM32H7 Series versus STM32F7 Series</td>
<td>31</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Example of tearing effect</td>
<td>32</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Independent layer management</td>
<td>32</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Hardware JPEG decoding</td>
<td>34</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Audio player module architecture</td>
<td>38</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Audio player module process</td>
<td>39</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Audio player module startup</td>
<td>39</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Audio player module hardware connectivity</td>
<td>40</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Video player module architecture</td>
<td>42</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Video player module process</td>
<td>42</td>
</tr>
<tr>
<td>Figure 29</td>
<td>rVideo player module startup</td>
<td>43</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Rocket game startup</td>
<td>44</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Rocket game ongoing</td>
<td>44</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Rocket game end</td>
<td>45</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Clock and weather startup</td>
<td>45</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Clock and weather module settings</td>
<td>46</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Clock and weather module skins</td>
<td>46</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Graphic effect main screen</td>
<td>47</td>
</tr>
<tr>
<td>Figure 37</td>
<td>System information main screen</td>
<td>47</td>
</tr>
<tr>
<td>Figure 38</td>
<td>TouchGFX demonstration</td>
<td>48</td>
</tr>
<tr>
<td>Figure 39</td>
<td>TouchGFX - Audio player module</td>
<td>48</td>
</tr>
<tr>
<td>Figure 40</td>
<td>TouchGFX - Video player module</td>
<td>49</td>
</tr>
<tr>
<td>Figure 41</td>
<td>TouchGFX - Time and calendar module</td>
<td>49</td>
</tr>
<tr>
<td>Figure 42</td>
<td>TouchGFX - Home control module</td>
<td>49</td>
</tr>
<tr>
<td>Figure 43</td>
<td>TouchGFX - Light effect module</td>
<td>50</td>
</tr>
<tr>
<td>Figure 44</td>
<td>TouchGFX - External hardware module</td>
<td>50</td>
</tr>
<tr>
<td>Figure 45</td>
<td>TouchGFX - Knight hit zombie game</td>
<td>51</td>
</tr>
<tr>
<td>Figure 46</td>
<td>TouchGFX - 2048 puzzle game</td>
<td>51</td>
</tr>
<tr>
<td>Figure 47</td>
<td>Embedded wizard demonstration</td>
<td>52</td>
</tr>
<tr>
<td>Figure 48</td>
<td>Embedded wizard - Video player module</td>
<td>52</td>
</tr>
</tbody>
</table>
Figure 49. Embedded wizard - Graphic effect module. .................................................. 53
Figure 50. Embedded wizard - TapTap plane module. .................................................. 53
Figure 51. Embedded wizard - Graphics accelerator module ....................................... 54
Figure 52. Embedded wizard - Waveform generator module ....................................... 54
Figure 53. Embedded wizard - Screen saver module .................................................... 55
STM32Cube overview

STM32Cube version 1.x includes the following elements:

- the STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards
- a comprehensive embedded software platform, delivered per series (such as STM32CubeH7 for STM32H7 Series):
  - the STM32CubeH7 HAL, STM32 abstraction layer embedded software ensuring maximized portability across the portfolio of STM32 32-bit Arm® Cortex®-based microcontrollers.
  - a consistent set of middleware components such as RTOS, FatFS, Graphics
  - all embedded software utilities coming with a full set of examples

Figure 1. STM32Cube block diagram
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 can be reused separately in a standalone application.

The full set of modules is managed by the kernel that 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 (SD card)
- 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, then the hardware resources needed during 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 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
199 /*
200  * $brief Start task
201  * $param argument: pointer that is passed to the thread function as start argument.
202  * $retval None
203  */
204 static void GUIThread(void const * argument)
205 {
206  /* Initialize Storage Units */
207  k_StorageInit();
208
209  /* Initialize GUI */
210  GUI_Init();
211
212  GUI_X_InitOS();
213
214  /* Enable Multibuffereing and set Layer0 as the default display layer */
215  WM_MULTIBUFF_Enable();
216  GUI_SetLayerVisEx (1, 0);
217  GUI_SelectLayer(0);
218
219  /* Show the main menu */
220  k_InitMenu();
221
222  /* Display immediately the Menu */
223  GUI_Execute();
224
225  /* GUI background Task */
226  while(1) {
227    osSemaphoreWait( LcdUpdateSemaphoreId , 1000 );
228    GUI_Execute();
229  }
230 }
```

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

```c
233 /**<
234  * $brief Timer callback (30 ms)
235  * $param n: Timer index
236  * $retval None
237 */
238 static void TimerCallback(void const *n)
239 {
240  k_TouchUpdate();
241  }
```

### 3.4 Kernel graphical aspect

The STM32Cube demonstration is built around the STemWin graphical library, based on SEGGER emWin one. STemWin is a professional graphical stack library, enabling graphical user interfaces (GUI) building up with any STM32, any LCD and any LCD controller, taking benefit from STM32 hardware accelerations.

The graphical aspect of the STM32Cube demonstration is divided into three main graphical components listed below:
• At the board reset, the splash screen is launched for some seconds. The splash screen can be skipped by a simple click on the screen, to launch the main menu of the three demonstrations.

**Figure 4. Splash screen**

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

**Figure 5. Demonstrations main menu**

Note: Only STemWin source code is available. The splash screen, TouchGFX and embedded wizard demonstrations are available only with the full binary file (STM32CubeDemo_STM32H743-Eval_VX.Y.Z_FULL.hex).

3.5 Kernel menu management

Note: **Important** - All the demonstration modules are described in this user manual. STemWin 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 to 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 to the startup function in the module structure; this is done when a WM_NOTIFICATION_RELEASED message arrives to the desktop callback with ID_ICONVIEW_MENU.

```c
/* Callback routine of desktop window.
    * @param pMsg: pointer to data structure of type WM_MESSAGE
    * @retval None
*/
static void _cbKK(WM_MESSAGE * pMsg) {
    ...
    switch (pMsg->msgID)
    {
    case WM_NOTIFY_PARENT:
        Id = WM_GetId(pMsg->hwndSrc);
        NCode = pMsg->wParam;
        switch (NCode)
        {
        case WM_NOTIFY:
            if (ID == ID_ICONVIEW_MENU)
            {
                sel = ST_AnimatedIconView_GetSel(pMsg->hwndSrc);
                if (sel < k_ModuleGetNumber())
                {
                    ST_AnimatedIconView_SetSel(pMsg->hwndSrc, -1);
                    if(module_active == 0)
                    {
                        module_pscp[sel]->module->startup(pMsg->hwnd, 0, 0);
                        if sel == 0
                        {
                            module_active = 1;
                        }
                        sel = 0;
                    }
                }
                else
                {
                    WM_InvalidRect((HWND)pMsg->hwndSrc);
                }
            }
            break;
        }
    
    case WM_NOTIFY_PARENT:
        Id = WM_GetId(pMsg->hwndSrc);
        NCode = pMsg->wParam;
        switch (NCode)
        {
        case WM_NOTIFY:
            if (ID == ID_ICONVIEW_MENU)
            {
                sel = ST_AnimatedIconView_GetSel(pMsg->hwndSrc);
                if (sel < k_ModuleGetNumber())
                {
                    ST_AnimatedIconView_SetSel(pMsg->hwndSrc, -1);
                    if(module_active == 0)
                    {
                        module_pscp[sel]->module->startup(pMsg->hwnd, 0, 0);
                        if sel == 0
                        {
                            module_active = 1;
                        }
                        sel = 0;
                    }
                }
                else
                {
                    WM_InvalidRect((HWND)pMsg->hwndSrc);
                }
            }
            break;
        }
```
3.6 Module manager

The modules are managed by the kernel, that is responsible of 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](image)

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.

```c
/* Initialise Modules function */
k_ModultInit();

/* Link modules */
k_ModulesAdd(audio_player_board);
k_ModulesAdd(video_player_board);
k_ModulesAdd(socket_game_board);
k_ModulesAdd(analog_clock_board);
k_ModulesAdd(graphic_effects_board);
```

A module is a set of function and data structures, that 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. Each module is identified by a unique identifier (UID). When two modules have the same UID, the kernel rejects the second one.

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

The module structure is defined as follows:
- **id**: unique module identifier
- **name**: pointer to 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 be saved, must be a 32-bit data and can be defined as a bit field structure.

```c
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 : 9;
        uint32_t reserved : 18;
    } b;
} AudioSettingsTypeDef;
```
Two kernel APIs are used to save or restore the structure from the RTC backup registers.

```c
void k_BkpSetParameter(uint32_t address, uint32_t data);
uint32_t k_BkpRestoreParameter(uint32_t address);
```

### 3.8 Storage units

The STM32Cube demonstration kernel offers a storage unit that may be used to retrieve audio and video media. The storage unit is initialized during the platform startup and available to all the modules during the STM32Cube demonstration run time.

![Figure 8. Available storage units](image)

The unit is accessible through the standard I/O operations offered by the FatFS used in the development platform. The SD card unit is identified as the unit 0 and available only if the SD card is connected on CN13. 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>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk_initialize</td>
<td>Initialize disk drive</td>
</tr>
<tr>
<td>disk_read</td>
<td>Interface function for a logical page read</td>
</tr>
<tr>
<td>disk_write</td>
<td>Interface function for a logical page write</td>
</tr>
<tr>
<td>disk_status</td>
<td>Interface function for testing if unit is ready</td>
</tr>
<tr>
<td>disk_ioctl</td>
<td>Control device dependent features</td>
</tr>
<tr>
<td>f_mount</td>
<td>Register / unregister a work area</td>
</tr>
<tr>
<td>f_open</td>
<td>Open / create a file</td>
</tr>
<tr>
<td>f_close</td>
<td>Close a file</td>
</tr>
<tr>
<td>f_read</td>
<td>Read a file</td>
</tr>
<tr>
<td>f_write</td>
<td>Write a file</td>
</tr>
<tr>
<td>f_lseek</td>
<td>Move read / write pointer, expand file size</td>
</tr>
<tr>
<td>f_truncate</td>
<td>Truncate file size</td>
</tr>
<tr>
<td>f_sync</td>
<td>Flush cached data</td>
</tr>
<tr>
<td>f_opendir</td>
<td>Open a directory</td>
</tr>
<tr>
<td>f_readdir</td>
<td>Read a directory item</td>
</tr>
</tbody>
</table>
For the FatFS file system, the page size is fixed to 512 bytes. The SD cards with higher page size are not supported.

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

**Table 1. File system interface functions (continued)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_getfree</td>
<td>Get free clusters</td>
</tr>
<tr>
<td>f_stat</td>
<td>Get a file status</td>
</tr>
<tr>
<td>f_mkdir</td>
<td>Create a directory</td>
</tr>
<tr>
<td>f_unlink</td>
<td>Remove a file or a directory</td>
</tr>
<tr>
<td>f_chmod</td>
<td>Change attribute</td>
</tr>
<tr>
<td>f_utime</td>
<td>Change timestamp</td>
</tr>
<tr>
<td>f_rename</td>
<td>Rename / move a file or a directory</td>
</tr>
<tr>
<td>f_mkfs</td>
<td>Create a file system on the drive</td>
</tr>
<tr>
<td>f_forward</td>
<td>Forward file data to the stream directly</td>
</tr>
<tr>
<td>f_chdir</td>
<td>Change current directory</td>
</tr>
<tr>
<td>f_chdrive</td>
<td>Change current drive</td>
</tr>
<tr>
<td>f_getcwd</td>
<td>Retrieve the current directory</td>
</tr>
<tr>
<td>f_gets</td>
<td>Read a string</td>
</tr>
<tr>
<td>f_putchar</td>
<td>Write a character</td>
</tr>
<tr>
<td>f_puts</td>
<td>Write a string</td>
</tr>
<tr>
<td>f_printf</td>
<td>Write a formatted string</td>
</tr>
</tbody>
</table>

**Figure 9. Software architecture**

![Software architecture diagram](MS4v47899V1)
The FatFS is mounted upon the mass storage to allow an 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 SPECIFIC_DEMO_ADDRESS 0x08100000
#define SPECIFIC_DEMO_SIGNATURE_A 0x5AA55AAA
#define SPECIFIC_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 sets then 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.

In the STM32Cube package, the demonstration sources are located in “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

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

3.12 Kernel core files

Table 3. Kernel core files list

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Main program file</td>
</tr>
<tr>
<td>stm32h7xx_it.c</td>
<td>Interrupt handlers for the application</td>
</tr>
<tr>
<td>k_bsp.c</td>
<td>Provides the kernel BSP functions</td>
</tr>
<tr>
<td>k_menu.c</td>
<td>Kernel menu and desktop manager</td>
</tr>
<tr>
<td>k_module.c</td>
<td>Modules manager</td>
</tr>
<tr>
<td>stm32h7xx_hal_timebase_tim.c</td>
<td>Use the hardware timer to configure the time base</td>
</tr>
<tr>
<td>k_rtc.c</td>
<td>RTC and backup manager</td>
</tr>
<tr>
<td>k_startup.c</td>
<td>Demonstration startup windowing process</td>
</tr>
<tr>
<td>k_storage</td>
<td>Storage units manager</td>
</tr>
<tr>
<td>startup_stm32h743xx.s</td>
<td>Startup file</td>
</tr>
<tr>
<td>cpu_utils.c</td>
<td>CPU load calculation utility</td>
</tr>
</tbody>
</table>
3.13 Hardware settings

The STM32Cube demonstration supports STM32H7 Series devices and runs on STM32H743I-EVAL demo board from STMicroelectronics.

![STM32H743-EVAL demonstration board](image)

1. Picture is not contractual.

Table 4. Jumpers for demonstration boards

<table>
<thead>
<tr>
<th>Board</th>
<th>Jumper</th>
<th>Position description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32H743I-EVAL</td>
<td>JP3</td>
<td>Status: not fitted&lt;br&gt;The Bootloader_BOOT is managed by pin 6 of connector CN2 (RS232 DSR signal) when JP3 is closed. This configuration is used for boot loader application only.</td>
</tr>
<tr>
<td></td>
<td>JP9</td>
<td>Status: fitted&lt;br&gt;Used to measure MCU current consumption manually</td>
</tr>
<tr>
<td></td>
<td>JP10</td>
<td>PSU position. For power supply jack (CN10)</td>
</tr>
</tbody>
</table>
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 main frame 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 an easy and quick creation of the module frame window and all its components.

Figure 13. GUIBuilder overview

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, that 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 basic module graphical appearance is created, 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.

```c
502  /**
503   * @brief callback for play button
504   * @param pMsg: pointer to data structure of type WM_MESSAGE
505   * @retval None
506  */
507 static void _chButton_play(WM_MESSAGE * pMsg) {
508     switch (pMsg->MsgID) {
509         case WM_PAINT:
510             _OnPaint_play(pMsg->hWin, PlayerSettings.b.pause);
511             break;
512         default:
513             /* The original callback */
514             BUTTON_Callback(pMsg);
515             break;
516         }
517     }
```

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

Of course the new callback must be associated to the graphical element (in our case the Play button) at the moment of its creation, like below.

```c
1579     hItem = BUTTON_CreateEx(112, 420, 40, 40, pMsg->hWin, WM_CF_SHOW, 0, ID_PLAY_BUTTON);
1580     WM_SetCallback(hItem, _chButton_play);
```

**Figure 14. Graphics customization**

<table>
<thead>
<tr>
<th>(not pressed)</th>
<th>(pressed)</th>
<th>(deactivated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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.

In the example above, cbDialog refers to the main module callback routine. Its general skeleton is structured as follows.

```c
/**
 * Brief: Module window Startup
 * @param hWin: pointer to the parent handle.
 * @param xpos: X position
 * @param ypos: Y position
 * @retval None
 */
static void Startup(HWND hWin, UINT xpos, UINT ypos)
{
    GUI_CreateDialogBox(ab DialogCreate, GUI_COUNTOF(ab DialogCreate), cbDialog, hWin, xpos, ypos)
    ...
}
```

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: define the behavior of each button).

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, 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.

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.

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

![Figure 16. SDRAM initialization](image)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD layer 0</td>
<td>0xD000 0000</td>
</tr>
<tr>
<td>LCD layer 1</td>
<td>0xD0000000 + (size of frame buffer * NUM_VSCREENS(1) * NUM_BUFFERS)(2)</td>
</tr>
</tbody>
</table>

1. NUM_VSCREENS: number of virtual screen defined in LCDConf.c file.
2. NUM_BUFFERS: number of multi buffer defined in LCDConf.c file.
5.3.2 Touchscreen configuration

The touchscreen is controlled by BSP TS driver that uses the exc7200 component.

**Figure 17. Touch screen configuration**

The touchscreen is initialized in 'k_BspInit' following the used screen resolution as shown in the code below.

```c
/* Brief: Initializes LEDs, SRAM, touch screen, CRC and SRAM. */
#define None

void k_BspInit(void) {
    ...}

/* Brief: Initialize the Touch screen */
BSP_TS_Init(440, 480);
...}

/* Brief: Read the coordinate of the point touched and assign their
value to the variables u32_TSCoordinate and u32_TSCoordinate */
#define None

void k_TouchUpdate(void) {
    static GUI_PID_STATE TS_State = {6, 6, 0, 0};
    BSP_TS_GetState((TS_StateTypeDef *) &st);
    ...}

GUI_TOUCHScreenStateEx(4TS_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.
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 frame buffer start address register of the display controller. Now it must be considered that a display is refreshed by the display controller approximately 60 times per second. After each period there is a vertical synchronization signal, known as 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.

```c
126  /**
127  * @brief Main program
128  * @param None
129  * @retval int
130  */
131  int main(void)
132  { /* Configure the MPU attributes as Write Through */
133  MPU_Config();
134  /* Instruction cache and data cache enable */
135  CPU_CACHE_Enable();
```
6.3 Multi layers feature

The windows can be placed in any layer or display. Drawing operations can be used 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.

6.4 Hardware acceleration

With the STM32H743I-EVAL demonstration, the hardware acceleration capabilities of the STM32H743xx / STM32H753xx 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 ensures 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. The drawing operation is then done by only one function call of the DMA2D.

- **Drawing of high-color bitmaps**
  If the layer works in the same mode as the high-color bitmap has its pixel data available, these bitmaps can be drawn by one function call of the DMA2D. The following function is used to set up such a function:
  
  ```c
  LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_DRAWBMP_16BPP, pFunc);
  ```

- **Filling operations**
  Setting up the function for filling operations:
  
  ```c
  LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_FILLRECT, pFunc);
  ```

- **Copy operations**
  Setting up the functions for copy operations used by the function GUI_CopyRect():
  
  ```c
  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:
  
  ```c
  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:
  
  ```c
  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:
  
  ```c
  GUI_SetFuncAlphaBlending(pFunc);
  ```

- **Drawing antialiased fonts**
  Setting up the function for mixing single foreground and background colors used when drawing transparent antialiased text:
  
  ```c
  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 hardware JPEG decoding is supported by the STM32F7 and STM32H7 Series devices. 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

![Image showing maximum frame rate comparison between STM32F7 at 200 MHz (38 fps) and STM32H7 at 400 MHz (up to 100 fps), highlighting a 2.7 times faster speed.]
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>
<thead>
<tr>
<th>Module</th>
<th>Code memory (bytes)</th>
<th>Data memory (bytes)</th>
<th>Constant memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>5688</td>
<td>26876</td>
<td>1318945</td>
</tr>
<tr>
<td>Video</td>
<td>5424</td>
<td>100228</td>
<td>231664</td>
</tr>
<tr>
<td>Graphic effect</td>
<td>2260</td>
<td>112</td>
<td>10041307</td>
</tr>
<tr>
<td>Clock &amp; weather</td>
<td>6032</td>
<td>844</td>
<td>3061328</td>
</tr>
<tr>
<td>Rocket games</td>
<td>2052</td>
<td>484</td>
<td>1715108</td>
</tr>
<tr>
<td>System info</td>
<td>1152</td>
<td>0</td>
<td>120164</td>
</tr>
</tbody>
</table>

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 an amount 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*

<table>
<thead>
<tr>
<th>Resolution</th>
<th>RAM usage (Kbytes)</th>
<th>RAM usage, size dependent (Kbytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>160x120</td>
<td>45.5</td>
<td>12.5</td>
</tr>
<tr>
<td>320x340</td>
<td>58.0</td>
<td>25.0</td>
</tr>
<tr>
<td>480x272</td>
<td>70.5</td>
<td>37.5</td>
</tr>
<tr>
<td>640x480</td>
<td>83.0</td>
<td>50.0</td>
</tr>
</tbody>
</table>
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 CPU used. Note that the listed values are the requirements of the basic functions of each module.

Table 8. MemoSTemWin components memory requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>ROM (Kbytes)</th>
<th>RAM (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows manager</td>
<td>6.2</td>
<td>2.5 K</td>
<td>Additional memory requirements of basic application when using the Windows manager</td>
</tr>
<tr>
<td>Memory devices</td>
<td>4.7</td>
<td>7 K</td>
<td>Additional memory requirements of basic application when using memory devices</td>
</tr>
<tr>
<td>Antialiasing</td>
<td>4.5</td>
<td>2 * LCD_XSIZE</td>
<td>Additional memory requirements for the antialiasing software item</td>
</tr>
<tr>
<td>Driver</td>
<td>2 to 8</td>
<td>20</td>
<td>The memory requirements of the driver depend on the configured driver and whether a data cache is used or not. With a data cache, the driver requires more RAM.</td>
</tr>
<tr>
<td>Multi layer</td>
<td>2 to 8</td>
<td>-</td>
<td>If working with a multi-layer or a multi-display configuration, additional memory is required for each additional layer, because each requires its own driver.</td>
</tr>
<tr>
<td>Core</td>
<td>5.2</td>
<td>80</td>
<td>Memory requirements of a typical application without using additional software items</td>
</tr>
<tr>
<td>JPEG</td>
<td>12</td>
<td>36 K</td>
<td>Basic routines for drawing JPEG files</td>
</tr>
<tr>
<td>GIF</td>
<td>3.3</td>
<td>17 K</td>
<td>Basic routines for drawing GIF files</td>
</tr>
<tr>
<td>Sprites</td>
<td>4.7</td>
<td>16</td>
<td>Routines for drawing sprites and cursors</td>
</tr>
<tr>
<td>Font</td>
<td>1 to 4</td>
<td>-</td>
<td>Depends on the font size to be used</td>
</tr>
</tbody>
</table>

Table 9. Widget memory requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>ROM (Kbytes)</th>
<th>RAM (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUTTON</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>CHECKBOX</td>
<td>1</td>
<td>52</td>
</tr>
<tr>
<td>DROPDOWN</td>
<td>1</td>
<td>52</td>
</tr>
<tr>
<td>EDIT</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>FRAMEWIN</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>
1. The listed memory requirements of the widgets contain the basic routines required for creating and drawing the widget. Depending on the specific widget, there are several additional functions available that are not listed in this table.

<table>
<thead>
<tr>
<th>Component</th>
<th>ROM (Kbytes)</th>
<th>RAM (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>GRAPH_DATA_XY</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>HEADER</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>LISTBOX</td>
<td>1</td>
<td>56</td>
</tr>
<tr>
<td>LISTVIEW</td>
<td>1</td>
<td>44</td>
</tr>
<tr>
<td>MENU</td>
<td>1</td>
<td>52</td>
</tr>
<tr>
<td>MULTEDIT</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>PROGBAR</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>RADIO BUTTON</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>SCROLLBAR</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>SLIDER</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>TEXT</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>CALENDAR</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 9. Widget memory requirements\(^{(1)}\) (continued)
8 Functional description of the demonstration modules

8.1 Audio player

8.1.1 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.

8.1.2 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
- Only 8-Kbyte RAM required for audio processing

*Note:* MP3 format not supported but may be easily added (separate demo flavor).

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

Figure 23. Audio player module architecture
8.1.4 Performance

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

![Figure 24. Audio player module process](image)

8.1.5 Process description

The audio player initialization is done in 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](image)
### 8.1.6 Hardware connectivity

Figure 26. Audio player module hardware connectivity

![Diagram showing hardware connectivity](image)

#### 8.1.7 Audio player module controls

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
</table>
| Play | ![Play icon]() | – Change the audio player state to "AUDIPLAYER_PLAY"
– Read the wave file from storage unit
– Set the frequency
– Start or resume the audio task
– Start playing audio stream from a data buffer using BSP_AUDIO_OUT_Play function in BSP audio driver
– Replace play button by pause button |
| Pause | ![Pause icon]() | – Suspend the audio task
– Pause the audio file stream
– Replace pause button by play button |
| Stop | ![Stop icon]() | – Close the wave file from storage unit
– Suspend the audio task
– Stop audio playing
– Change the audio player state to "AUDIPLAYER_STOP" |
8.2 Video player

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

8.2.2 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 2 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**

8.2.3 Performance

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

**Figure 28. Video player module process**
8.2.4 Functional description

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

![Video player module startup](image)

**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>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play/Pause</td>
<td></td>
<td>– Read the AVI file from storage unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Start playing audio stream</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Replace play/pause button by pause/play button</td>
</tr>
<tr>
<td>Previous</td>
<td></td>
<td>– Point to the previous avi file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Stop video playing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Start playing the previous avi file</td>
</tr>
<tr>
<td>Next</td>
<td></td>
<td>– Point to the previous avi file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Stop video playing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Start playing the previous avi file</td>
</tr>
<tr>
<td>HW JPEG</td>
<td></td>
<td>– Enable and disable the JPEG hardware decoding</td>
</tr>
<tr>
<td>ChromART</td>
<td></td>
<td>– Enable and disable the use of ChromART for YCbCr to ARGB conversion</td>
</tr>
<tr>
<td>Exit</td>
<td></td>
<td>– Close video player module</td>
</tr>
</tbody>
</table>
8.3 Rocket game

8.3.1 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.

8.3.2 Functional description

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

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

Figure 32. Rocket game end

8.4 Clock and weather

8.4.1 Overview

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

Note: Only 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 Settings button to choose the clock and change the skin.

**Figure 34. Clock and weather module settings**

![Figure 34](image)

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

**Figure 35. Clock and weather module skins**

![Figure 35](image)

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

### 8.5 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
- sharpening filter
- raising filter
- motion blur filter

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

8.6 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.
8.7 TouchGFX demonstration

8.7.1 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 38. TouchGFX demonstration

8.7.2 Audio player module

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.

Figure 39. TouchGFX - Audio player module
8.7.3 Video player module

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.

Figure 40. TouchGFX - Video player module

8.7.4 Time and calendar module

The time and calendar module allows the user to show and adjust the time and date by changing the real-time configuration (RTC).

Figure 41. TouchGFX - Time and calendar module
8.7.5 **Home control module**

The home control module allows the user to control home and shows vivid graphs and access control.

*Figure 42. TouchGFX - Home control module*

8.7.6 **Light effect module**

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

*Figure 43. TouchGFX - Light effect module*
8.7.7 **External hardware module**

The external hardware module allows the graphic control via the potentiometer and the screen brightness adjustment.

**Figure 44. TouchGFX - External hardware module**

8.7.8 **Knight hit zombie game**

The knight hit zombie game shows the graphic performance of the Chrom-ART Accelerator and TouchGFX graphical stack.

**Figure 45. TouchGFX - Knight hit zombie game**

8.7.9 **2048 puzzle game**

The 2048 puzzle game shows high-quality graphics and smooth animations.

**Figure 46. TouchGFX - 2048 puzzle game**
8.8  Embedded wizard demonstration

8.8.1  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

---

8.8.2  Video player module

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.avi
- video1.avi
- video2.avi
- video3.avi
8.8.3 Graphic effect module

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 on the middle of top screen.

Figure 49. Embedded wizard - Graphic effect module

8.8.4 TapTap plane module

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 showcase the processing capabilities of 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 50. Embedded wizard - TapTap plane module
8.8.5 Graphics accelerator module

The graphic accelerator module is used to showcase 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 51. Embedded wizard - Graphics accelerator module

8.8.6 Waveform generator module

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

Figure 52. Embedded wizard - Waveform generator module
8.8.7 Screen saver module

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 showcase the CPU capabilities.

Figure 53. Embedded wizard - Screen saver module
9 Revision history

Table 12. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-Dec-2017</td>
<td>1</td>
<td>Initial version</td>
</tr>
</tbody>
</table>
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. 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.

© 2017 STMicroelectronics – All rights reserved