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 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 and STM32H747I DISC1 boards.

Pictures are not contractual.
# 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 ......................................... 13
   3.5 Kernel menu management ....................................... 14
   3.6 Module manager .................................................. 15
   3.7 Backup and settings configuration .............................. 17
   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
## Performance

6.1 CPU cache ................................................. 31
6.2 Multi buffering features ................................. 32
6.3 Multi layers feature ........................................ 33
6.4 Hardware acceleration .................................... 33
6.5 Hardware JPEG Decoding ................................. 35

## Footprint

7.1 STemWin features resources ............................. 36
  7.1.1 JPEG decoder .......................................... 36
  7.1.2 GUI components ....................................... 37

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

8.1 STemWin .................................................. 39
  8.1.1 Audio player ........................................... 39
  8.1.2 Video player ........................................... 44
  8.1.3 Rocket game ........................................... 47
  8.1.4 Clock and weather ..................................... 48
  8.1.5 Graphic effect ......................................... 49
  8.1.6 Dual core module ...................................... 50
  8.1.7 System information ..................................... 50

8.2 TouchGFX demonstration ................................. 51
  Overview ..................................................... 51
  8.2.1 Audio player module .................................. 51
  8.2.2 Graphics effect ....................................... 52
  8.2.3 Video player module .................................. 52
  8.2.4 Time and calendar module ............................ 53
  8.2.5 Home control module .................................. 54
  8.2.6 Light effect module .................................... 56
  8.2.7 External hardware module ............................ 57
  8.2.8 Bird eat coin ........................................... 57
  8.2.9 Knight hit zombie game ............................... 58
  8.2.10 2048 Puzzle game .................................... 58

8.3 Embedded wizard demonstration ......................... 60
## 8.3 Overview of the STM32H743I-EVAL demonstration modules

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.1</td>
<td>Overview</td>
<td>60</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Video player module (only H743I-EVAL)</td>
<td>60</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Graphic effect module (only H743I-EVAL)</td>
<td>61</td>
</tr>
<tr>
<td>8.3.4</td>
<td>TapTap plane module (only H743I-EVAL)</td>
<td>61</td>
</tr>
<tr>
<td>8.3.5</td>
<td>Graphics accelerator module</td>
<td>62</td>
</tr>
<tr>
<td>8.3.6</td>
<td>Waveform generator module</td>
<td>62</td>
</tr>
<tr>
<td>8.3.7</td>
<td>Screen saver module (only H743I-EVAL)</td>
<td>63</td>
</tr>
<tr>
<td>8.3.8</td>
<td>Charts demonstration</td>
<td>63</td>
</tr>
<tr>
<td>8.3.9</td>
<td>Climate cabinet</td>
<td>64</td>
</tr>
<tr>
<td>8.3.10</td>
<td>Brick game</td>
<td>64</td>
</tr>
<tr>
<td>8.3.11</td>
<td>Fitness tracker</td>
<td>65</td>
</tr>
<tr>
<td>8.3.12</td>
<td>Paper cutter</td>
<td>65</td>
</tr>
<tr>
<td>8.3.13</td>
<td>Washing machine</td>
<td>66</td>
</tr>
</tbody>
</table>

## 9 Functional description of the STM32H745I-DISCO demonstration modules

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Main graphical interface</td>
<td>67</td>
</tr>
<tr>
<td>9.2</td>
<td>System information</td>
<td>67</td>
</tr>
<tr>
<td>9.3</td>
<td>EEMBC® CoreMark®</td>
<td>68</td>
</tr>
<tr>
<td>9.4</td>
<td>Oscilloscope and signals generator</td>
<td>69</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Overview</td>
<td>69</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Features</td>
<td>70</td>
</tr>
<tr>
<td>9.4.3</td>
<td>Hardware and software setup</td>
<td>71</td>
</tr>
<tr>
<td>9.4.4</td>
<td>How to use</td>
<td>72</td>
</tr>
</tbody>
</table>

## Revision history

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
</tr>
</tbody>
</table>
List of tables

Table 1. File system interface functions .................................................. 18
Table 2. Kernel components list ................................................................. 22
Table 3. Kernel core files list ................................................................. 22
Table 4. Jumpers for demonstration boards ............................................... 23
Table 5. LCD frame buffer locations ...................................................... 28
Table 6. Modules footprint ................................................................. 36
Table 7. RAM requirements for some JPEG resolutions ............................ 36
Table 8. MemoSTemWin components memory requirements .................. 37
Table 9. Widget memory requirements .................................................... 37
Table 10. Audio player module controls .................................................. 43
Table 11. Video player module controls .................................................. 46
Table 12. Document revision history ......................................................... 75
List of figures

Figure 1. STM32Cube block diagram ................................. 8
Figure 2. STM32CubeH7 demonstration architecture .................. 9
Figure 3. Kernel components and services .......................... 10
Figure 4. Splash screen .............................................. 13
Figure 5. Demonstrations main menu ............................... 13
Figure 6. STemWin demonstration main menu ......................... 14
Figure 7. Functionalities and properties of modules ................. 15
Figure 8. Available storage units ................................... 17
Figure 9. Software architecture .................................... 19
Figure 10. Demonstration memory mapping ........................ 19
Figure 11. Demonstration folder structure .......................... 21
Figure 12. STM32Cube demonstration boards ......................... 23
Figure 13. GUIBuilder overview .................................. 24
Figure 14. Graphics customization ................................ 25
Figure 15. LCDConf.c location ..................................... 27
Figure 16. SDRAM initialization .................................... 28
Figure 17. Touch screen configuration ............................... 29
Figure 18. STM32H7 Series system architecture .................... 31
Figure 19. Performance of STM32H7 Series versus STM32F7 Series 32
Figure 20. Example of tearing effect ............................... 33
Figure 21. Independent layer management .......................... 33
Figure 22. Hardware JPEG decoding ................................ 35
Figure 23. Audio player module architecture ....................... 40
Figure 24. Audio player module process ........................... 40
Figure 25. Audio player module startup ........................... 41
Figure 26. Connectivity of audio player module hardware .......... 42
Figure 27. Video player module architecture ....................... 44
Figure 28. Video player module process ........................... 45
Figure 29. Video player module startup ........................... 45
Figure 30. Rocket game startup ................................... 47
Figure 31. Rocket game ongoing ................................... 47
Figure 32. Rocket game end ........................................ 48
Figure 33. Clock and weather startup .............................. 48
Figure 34. Clock and weather module settings ..................... 48
Figure 35. Clock and weather module skins ......................... 49
Figure 36. Graphic effect main screen ............................. 49
Figure 37. Dual core module ....................................... 50
Figure 38. System information main screen ......................... 50
Figure 39. TouchGFX demonstration ............................... 51
Figure 40. TouchGFX - Audio player module ....................... 51
Figure 41. The three audio menus, playlist (left), directories (middle) and equalizer (right) .......... 52
Figure 42. Graphics effect module .................................. 52
Figure 43. TouchGFX - Video player module ....................... 53
Figure 44. The three video player menus, video player (left), directory (middle) and playlist (right) 53
Figure 45. TouchGFX - Time and calendar module .................. 54
Figure 46. The three settings menus, watch face (left), set time (middle) and set date (right) .......... 54
Figure 47. TouchGFX - Home control module ....................... 55
Figure 48. Menu to set the blinds, like light and security but options only to tune on and off .......... 55
List of figures

Figure 49. TouchGFX - Light effect module ................................................................. 56
Figure 50. TouchGFX - External hardware module ..................................................... 57
Figure 51. Bird eat coin game ................................................................................. 58
Figure 52. TouchGFX - Knight hit zombie game ......................................................... 58
Figure 53. 2048 Puzzle game module ..................................................................... 59
Figure 54. The How To Play modal window ............................................................... 59
Figure 55. Embedded wizard demonstration ............................................................. 60
Figure 56. Embedded wizard - Video player module ............................................... 60
Figure 57. Embedded wizard - Graphic effect module .............................................. 61
Figure 58. Embedded wizard - TapTap plane module .............................................. 61
Figure 59. Embedded wizard - Graphics accelerator module ................................... 62
Figure 60. Embedded wizard - Waveform generator module ................................... 62
Figure 61. Embedded wizard - Screen saver module .............................................. 63
Figure 62. Charts demonstration ............................................................................ 63
Figure 63. Climate cabinet ...................................................................................... 64
Figure 64. Brick game ............................................................................................ 64
Figure 65. Fitness tracker ....................................................................................... 65
Figure 66. Paper cutter ........................................................................................... 65
Figure 67. Washing machine ................................................................................. 66
Figure 68. Main menu ............................................................................................ 67
Figure 69. System information module .................................................................. 68
Figure 70. EEMBC® CoreMark® application .......................................................... 68
Figure 71. Oscilloscope and signals generator application ....................................... 69
Figure 72. Oscilloscope and signals generator overview .......................................... 71
Figure 73. Signal generator to oscilloscope connection ............................................ 72
Figure 74. External signal to oscilloscope connection ............................................. 72
Figure 75. PC oscilloscope screen ......................................................................... 73
Figure 76. STM32H745I-DISCO board screen ......................................................... 73
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 & MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeH7 for the STM32H7 Series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the portfolio of STM32 32-bit Arm 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 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
void GUIThread(void const *argument) {
    /* Initialize Storage Units */
    k_StorageInit();
    /* Initialize GUI */
    GUI_Init();
    GUI_X_Init();
    // Enable Multibuffering and set Layer0 as the default display layer */
    NM_MULTIBUF_Enable();
    // Show the main menu */
    k_InitMenu();
    // Display immediately the Menu */
    GUI_Execute();
    /* GUI background task */
    while(1) {
        osSemaphoreWait(GUIUpdateSemaphoreId, 1000);
        GUI_Execute();
    }
}
```

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

```c
void TimerCallback(void const *n) {
    k_TouchUpdate();
}
```

**Note:** For the STM32H747I EVAL and 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](image)

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

![Figure 5. Demonstrations main menu](image)

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 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 performed when a `WM_NOTIFICATION_RELEASED` message arrives to the desktop callback with `ID_ICONVIEW_MENU`.

```c
/**
 * @brief Callback routine of desktop window.
 * @param pMsg: pointer to data structure of type WM_MESSAGE
 * @param hMenu:
 */
static void _callback(WM_MESSAGE * pMsg)
{
...
switch (pMsg->msgId)
{
    case WM_NOTIFY_PARENT:
        if (pMsg->message == WM_NOTIFY_PARENT &&
            pMsg->message == WM_ICONVIEW_MENU)
        {
            sel = ST_AutomatedIconView_GetSel(pMsg->hWinSrc);
            if (sel < k_ModuleSetNumber())
                ST_AutomatedIconView_GetSel(pMsg->hWinSrc, -1);
            if (module_active == 0)
                module_pconv[sel]->module->startup(pMsg->hWin, 0, 0);
            if (module_active == 1)
                module_pconv[sel]->module->startup(pMsg->hWin, 0, 0);
            module_active = 1;
        }
        else
            break;
}
```

### 3.6 Module manager

The kernel manages the modules. It 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**

![Diagram showing the functionalities and properties of modules](MSv4788FV1)
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_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 function 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.

```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) (GUIHWND, uint16_t, uint16_t);
    void (*DirectOpen) (char *);
} k_ModuleIcon_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 save must be a 32-bit data and can be defined as a bit field structure.

```c
typedef union
{
    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;
    }
} AudioSettingsTypeDef;
```

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

```c
void k_BkgSaveParameter(uint32_t address, uint32_t data);
uint32_t k_BkgRestoreParameter(uint32_t address);
```

3.8 Storage units

The STM32Cube demonstration kernel offers a storage unit that is usable 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](image)

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.
In 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.
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 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.

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

<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>krtc.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_stm32h7xxxx.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, STM32H747I EVAL, STM32H747I DISCO and STM32H747I DISC1 boards from STMicroelectronics.

Figure 12. STM32Cube demonstration boards

1. Pictures are not contractual.

<table>
<thead>
<tr>
<th>Board</th>
<th>Jumper</th>
<th>Position description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32H743I-EVAL and STM32H747I-EVAL</td>
<td>JP3</td>
<td>Status: not fitted&lt;br&gt;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.</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>Position: PSU position. For power supply jack (CN10)</td>
</tr>
<tr>
<td>STM32H747I-DISCO and STM32H747I-DISC1</td>
<td>JP3</td>
<td>Status: fitted&lt;br&gt;Used to measure MCU current consumption manually</td>
</tr>
<tr>
<td></td>
<td>JP6</td>
<td>Position: STllk (STLINK)&lt;br&gt;Used to select power supply source.</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 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 an easy and quick creation of the module frame window and all its components.

![GUIBuilder overview](image)

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 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.

```c
/*
 * @brief callback for play button
 * @param pMsg: pointer to data structure of type WM_MESSAGE
 * @retval None
 */
static void _chButton_play(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
    case WM_PAINT:
    _OnPaint_play(pMsg->hwnd, 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 to the graphical element (in our case the Play button) when it is created, like below.

```c
hItem = BUTTON_CreateEx(112, 420, 40, 40, pMsg->hwnd, WM_CF_SHOW, 0, ID_PLAY_BUTTON);
WM_SetCallback(hItem, _chButton_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.

```c
1469 /**
1470 * Module window Startup
1471 * @param hWnd: pointer to the parent handle.
1472 * @param xpos: X position
1473 * @param ypos: Y position
1474 * @retval None
1475 */
1476 static void Startup(HWND hWnd, UINT16_t xpos, UINT16_t ypos)
1477 {
1478     GUI_CreateDialogBox(aDialogCreate, GUI_COUNTOF(aDialogCreate), _cbDialog, hWnd, xpos, ypos)
1479     ...}
```

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

```c
531 /**
532 * Callback routine of the dialog
533 * @param pMsg: pointer to data structure of type WM_MESSAGE
534 * @retval None
535 */
536 static void _cbDialog(WM_MESSAGE * pMsg)
537 {
538     case WM_INIT_DIALOG:
539         /* Initialize graphical elements and restore backup parameters if any */
540         break;
541     case WM_NOTIFY_PARENT:
542         NCode = pMsg->hWinSrc;
543         case ID_BUTTON:
544             switch (NCode) {
545                 case WM_NOTIFICATION_RELEASED:
546                     /* Operation associated to the button */
547             }
548         ...}
```

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, 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.

![Figure 16. SDRAM initialization](MSv47834V1)

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

**Table 5. LCD frame buffer locations**

<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>0xD00000000 + (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 the BSP TS driver, which uses the exc7200 component in case of the STM32H743I-EVAL board, and the ft6x06 component for the STM32H747I EVAL and STM32H747I DISCO boards.

Figure 17. Touch screen configuration
The touch screen is initialized in `k_BspInit` following the used screen resolution as shown in the code below.

```c
71    /*
72    * @brief    Initializes LEDs, SDRAM, touch screen, CRC and SRAM.
73    * @param    None
74    * @retval   None
75    */
76    void k_BspInit(void)
77    {
78        /* Initialize the Touch screen */
79        BSP_TS_Init(640, 480);
80        /* */
81    }
82
83    /*
84    * @brief    Read the coordinate of the point touched and assign their
85    * value to the variables uTS_TSCoordinate and uTS_TSICoordinate
86    * @param    None
87    * @retval   None
88    */
89    void k_TouchUpdate(void)
90    {
91        static GUI_PID_STATE TS_State = {0, 0, 0};
92        BSP_TS_GetState((TS_StateTypeDef *)state);
93        /* */
94        GUI_TOUCH_StoreStateEx(uTS_State);
95        /* */
96    }
97        /* */
98    }
99    /* */
100    }
101    /* */
102    }
103    /* */
104    */
```
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

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

```c
126  /**
127  * @brief Main program
128  * @paroz Done
129  * @retval int
130  */
131  int main(void)
132  {  
133  /* Configure the MPU attributes as Write Through */
134  MPU_Config();
135  
136  /* Instruction cache and data cache enable */
137  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 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.
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.

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:

  ```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 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

![Diagram showing maximum frame rate comparison between STM32F7 at 200 MHz (38 fps) and STM32H7 at 400 MHz (Up to 100 fps), with an x 2.7 faster notation.]
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)

<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. Memory requirements of STemWin components**

<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 the presence of a data cache. 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, 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>MULTIEDIT</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 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.
Performance

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

**Figure 24. Audio player module process**

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
Hardware connectivity

Figure 26. Connectivity of audio player module hardware
## Audio player module controls

### Table 10. Audio player module controls

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
</table>
| **Play**           | ![play]  | – Change the audio player state to "AUDIOPLAYER_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] | – Suspend the audio task  
                   |           |  
                   |           | – Pause the audio file stream  
                   |           |  
                   |           | – Replace pause button by play button  
| **Stop**           | ![stop]  | – Close the wave file from storage unit  
                   |           |  
                   |           | – Suspend the audio task  
                   |           |  
                   |           | – Stop audio playing  
                   |           |  
                   |           | – Change the audio player state to "AUDIOPLAYER_STOP"  
| **Previous**       | ![previous] | – Point to the previous wave file  
                   |           |  
                   |           | – Stop audio playing  
                   |           |  
                   |           | – Start playing the previous wave file if play button is pressed  
| **Next**           | ![next]  | – Point to the next wave file  
                   |           |  
                   |           | – Stop audio playing  
                   |           |  
                   |           | – Start playing the next wave file if play button is pressed  
| **Add file to playlist** | ![playlist] | – Open playlist window  
                   |           |  
                   |           | – Add audio file from SD Card or choose audio file to play  
| **Visualization**  | ![visualization] | Disable or enable the display of graphic animation based on real time PCM audio samples  
| **Menu**           | ![menu]  | Close audio player module  
| **Volume**         |          | – Volume up  
                   |           |  
                   |           | – Volume Down  
| **Time**           | ![time]  | – Move the Music time forward  
                   |           |  
                   |           | – Move the Music time back  

*STI*
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 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.
Performance

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

![Figure 28. Video player module process](image)

Functional description

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

![Figure 29. 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 11. Video player module controls**

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
</table>
| **Play/Pause** | ![Play/Pause] | – Read the AVI file from storage unit  
– Start playing audio stream  
– Replace play/pause button by pause/play button |
| **Previous** | ![Previous] | – Point to the previous AVI file  
– Stop video playing  
– Start playing the previous AVI file |
| **Next** | ![Next] | – Point to the previous AVI file  
– Stop video playing  
– Start playing the previous AVI file |
| **HW JPEG** | ![HW JPEG] | – Enable and disable the JPEG hardware decoding |
| **ChromART** | ![ChromART] | – Enable and disable the use of Chrom-ART for YCbCr to ARGB conversion |
| **Exit** | ![Exit] | – Close video player module |
| **Volume** | ![Volume] | – Volume up  
– Volume down |
| | ![Volume] | – 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**

   ![Rocket game startup](image1)

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**

   ![Rocket game ongoing](image2)
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 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 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 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
- Sharpening filter
- Raising filter
- Motion blur filter

The CPU load metrics are displayed on the middle of the top screen.
8.1.6 Dual core module

Overview

The dual core module demonstrates the platform's dual core capability while decoding four video files and rendering fractal bitmap.

Checking the check box offloads the fractal rendering on the second core (CM4). This should decrease the CM7 core loading.

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.
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.

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.
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.

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.
8.2.4 Time and calendar module

Overview

The time and calendar lets the user change the time and date which is handle 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 displays the time and 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 the Time and calendar module is exited and the changes that has not been confirmed by pressing done is 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 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
8.2.7 External hardware module

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

In the module are the user 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

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](image)

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, in 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](image)

8.2.10 2048 Puzzle game

Overview

The 2048 game module, is the classic game 2048 created with TouchGFX, which show cases 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](image)

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 windows, which describes the rules and the goal of the game.

![Figure 54. The How To Play modal window](image)
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](image)

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](image)
### 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
- Sharpening filter
- Raising filter
- Motion blur filter

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

![Figure 57. Embedded wizard - Graphic effect module](image)

### 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 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](image)
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

8.3.6 Waveform generator module

The waveform generator module displays the possibility to emulate waveform generation and signal frequency display.
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 axis (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 a main menu that is usable 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 demonstration firmware version.
9.3 EEMBC® CoreMark®

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 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

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
- 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 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, Cortex-M4 continue 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.
9.4.3 Hardware and software setup

1. Install the PC GUI application on your 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
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 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

**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.
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.

If there is no user action on the touchscreen during 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 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 and the internal FLASH 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
# 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>
<tr>
<td>12-Jun-2019</td>
<td>2</td>
<td>Updated <em>Introduction</em>, <em>Section 8</em> to add new supported boards and applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added <em>Section 9</em> specific to STM32H745I-DISCO</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. 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.

© 2019 STMicroelectronics – All rights reserved