UM1919
User manual

STM32CubeL4 demonstration firmware for 32L476GDISCOVERY discovery kit

Introduction

The STMCube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL4 for STM32L4 Series)
  - The STM32CubeL4 HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, STMTouch and FatFs
  - All embedded software utilities coming with a full set of examples

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

With a LCD-glass display, a microphone, a joystick, ST-LINK/V2 debugger/programmer and microcontrollers from the STM32L1 and STM32L4 Series, this discovery kit is ideal to evaluate STM32 ultra-low-power solutions and audio capabilities.

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

The STM32CubeL4 demonstration firmware supports STM32L476xx devices and runs on 32L476GDISCOVERY discovery kit.

Contents

List of tables

List of figures

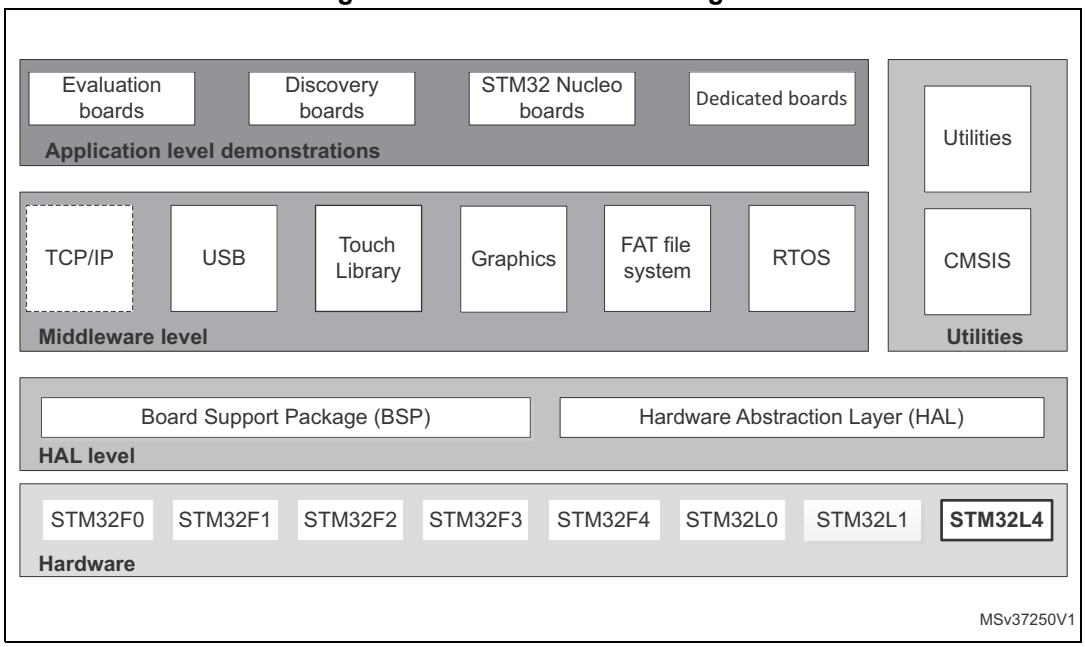
1 STM32Cube overview

The STMCube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL4 for STM32L4 Series)
  - The STM32CubeL4 HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, graphics
  - All embedded software utilities coming with a full set of examples.

Figure 1. STM32Cube block diagram



2 Getting started with demonstration

2.1 Hardware requirements

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

- STM32L476G discovery board (*Figure 2* and *Figure 3*) (refer to UM1879 for the discovery kit description)
- One “USB type A to Mini-B” cable to power up the discovery board from the USB ST-LINK (USB connector CN1) and to run in USB-powered mode
- One CR2032 battery to run in battery mode.

The STM32L476G discovery kit helps to discover the ultra-low-power features and audio capabilities of the STM32L4 Series. It offers everything required for beginners and experienced users to get stared quickly and develop applications easily.

Based on an STM32L476VGT6 MCU, it includes an ST-LINK/V2-1 embedded debug tool interface, Idd current measurement, Quad-SPI Flash memory, an audio codec with 3.5mm connector, LCD segment display (8x40), LEDs, a joystick and a USB mini-B connector.

Figure 2. STM32L476G discovery board (top view)

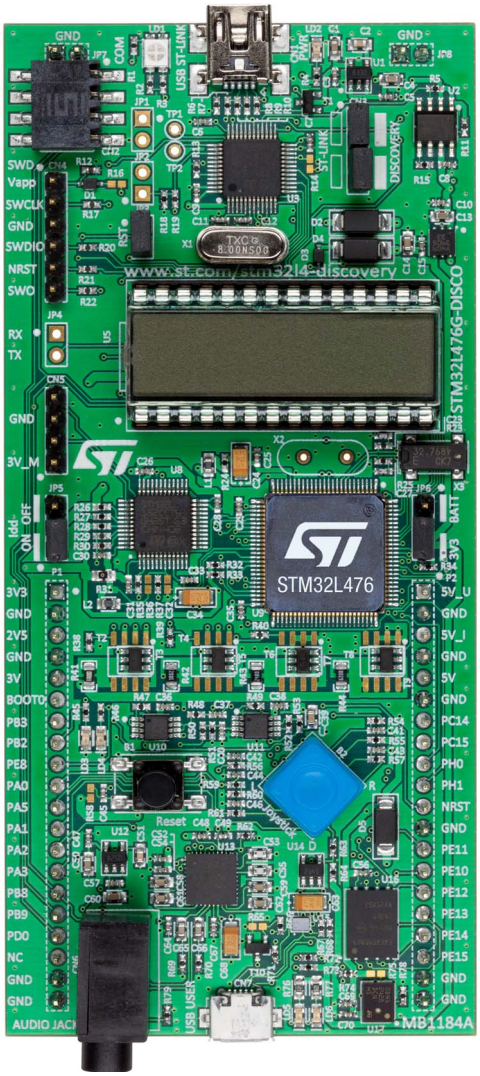
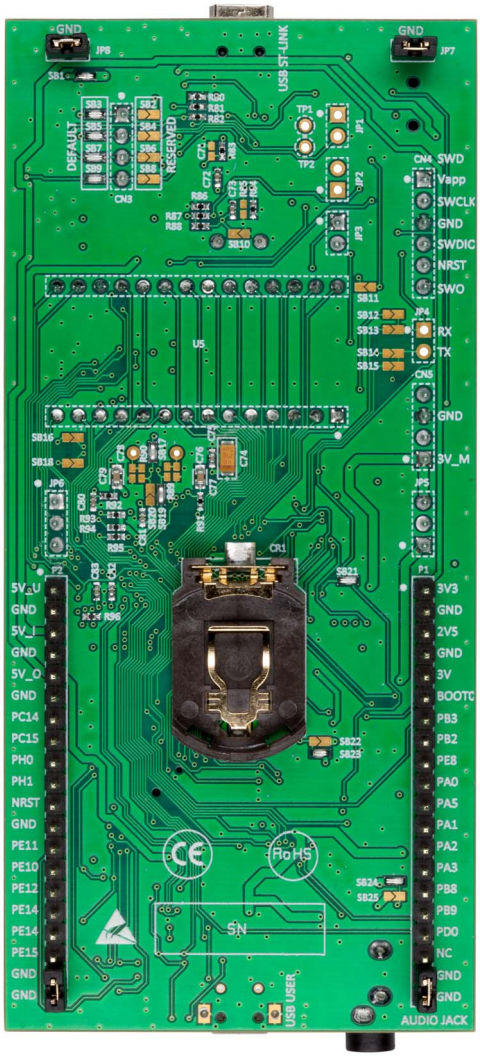


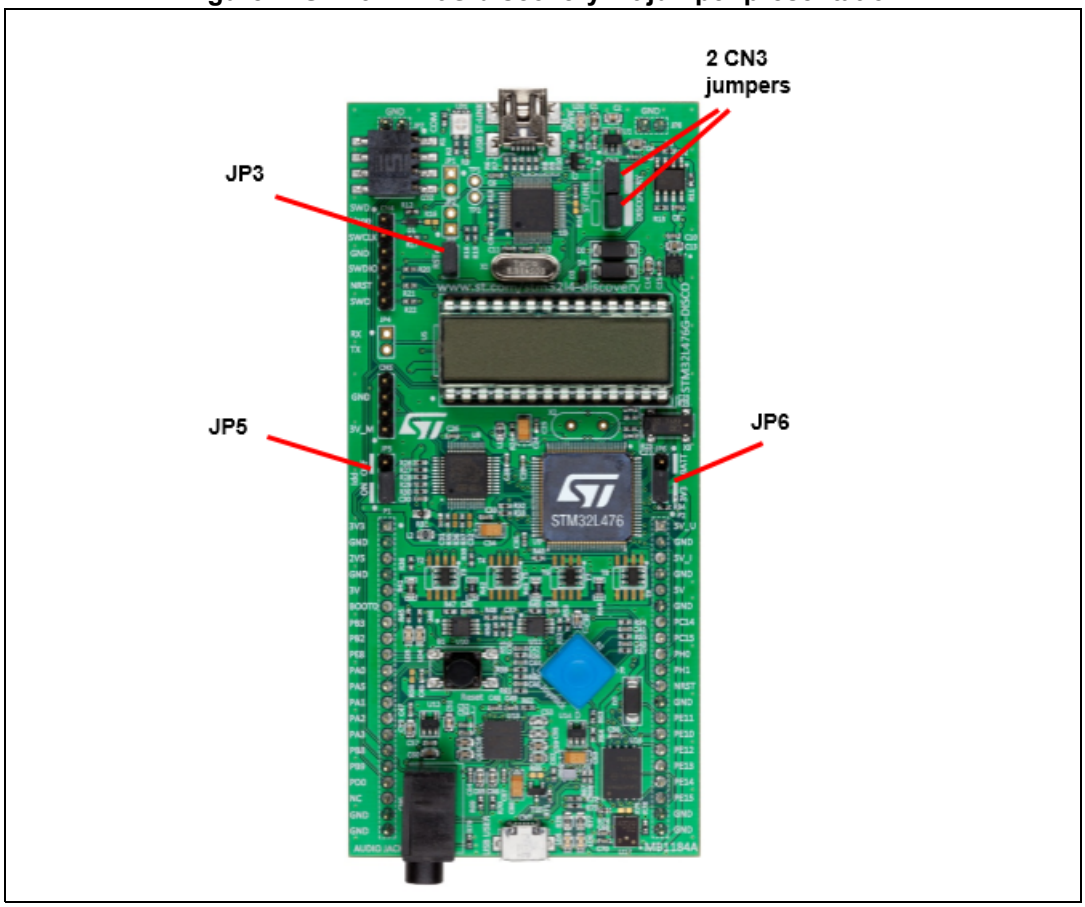
Figure 3. STM32L476G discovery board (bottom view)



2.2 Hardware configuration

The STM32Cube demonstration supports STM32L476xx devices and runs on STM32L476G-DISCO demonstration board from STMicroelectronics. Several jumpers are used to configure the hardware mode of operation as shown in *Figure 4*.

Figure 4. STM32L476G discovery kit jumper presentation



2.3 Power supply modes

The demonstration firmware provides two modes of operation depending on the power supply mode. The richest power supply mode is the USB-powered where all the applications can run at a maximum clock speed whereas an alternative mode consists in running from the battery. The battery mode allows to demonstrate the low-power consumption of MCU but also the HW itself. The demonstration firmware implements thus some clock reduction mechanisms, enters automatically in low-power modes in case of inactivity and wake-up in case of end-user interaction with the joystick.

2.3.1 USB supply

The following jumper setup is mandatory for running the demonstration powered by the USB (ST-LINK):

- Jumpers RST (JP3) / ST-LINK (CN3) present
- Jumper JP5 on IDD
- Jumper JP6 on 3V3

2.3.2 Battery supply

The following jumper setup is mandatory for running the demonstration powered by the battery (see *Figure 4*):

- Jumpers RST (JP3) / ST-LINK (CN3) removed
- Jumper JP5 on IDD
- Jumper JP6 on BATT

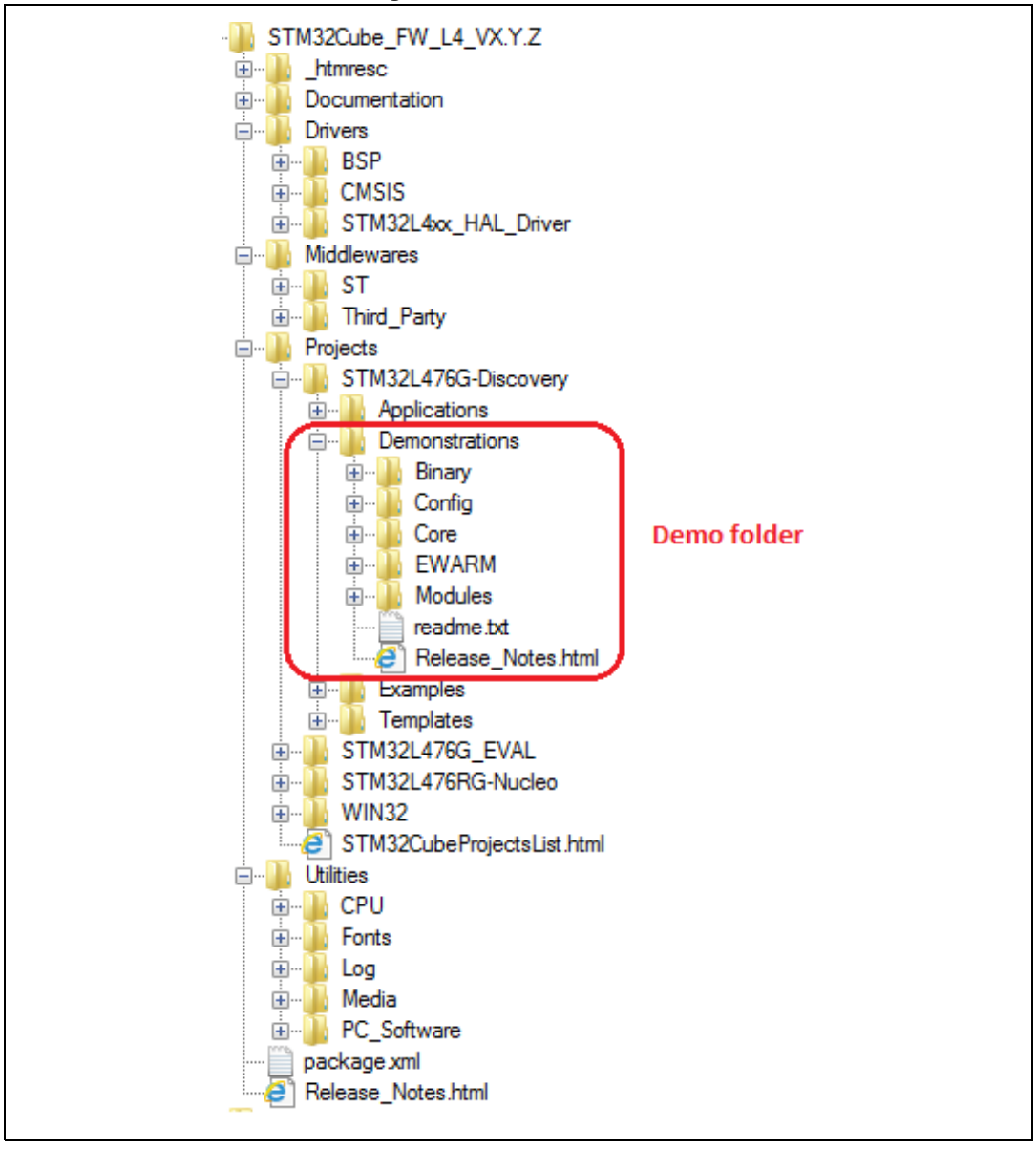
And of course it insures that a CR2032 battery is present at the rear of the board.

3 Demonstration firmware package

3.1 Demonstration repository

The STM32CubeL4 demonstration firmware for 32L476GDISCOVERY discovery kit is provided within the STM32CubeL4 firmware package as shown in *Figure 5*.

Figure 5. Folder structure



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

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

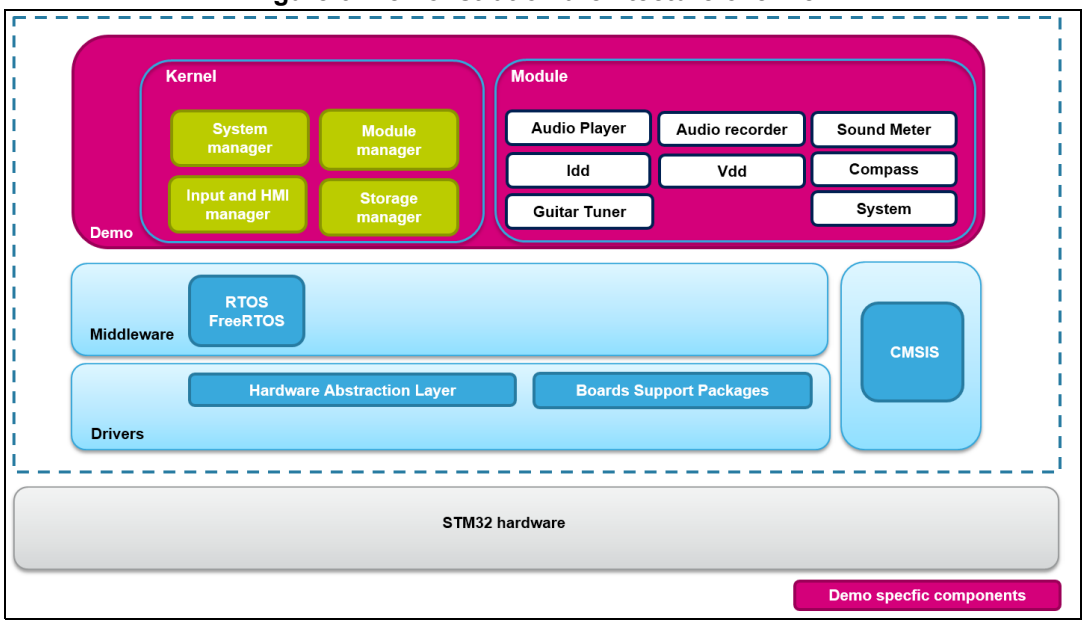
3.2 Demonstration architecture overview

The STM32CubeL4 demonstration firmware for 32L476GDISCOVERY discovery kit is composed of a central kernel based on a set of firmware and hardware services offered by the STM32Cube middleware, discovery board drivers and a set of modules mounted on the kernel and built in a modular architecture. Each module can be reused separately in a standalone application. The full set of modules is managed by the Kernel which provides access to all common resources and facilitates the addition of new modules as shown in *Figure 6*.

Each module should provide the following functionalities and properties:

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

Figure 6. Demonstration architecture overview



3.2.1 Kernel core files

Table 1 lists the kernel core files covering the startup, interrupts management and kernel core services.

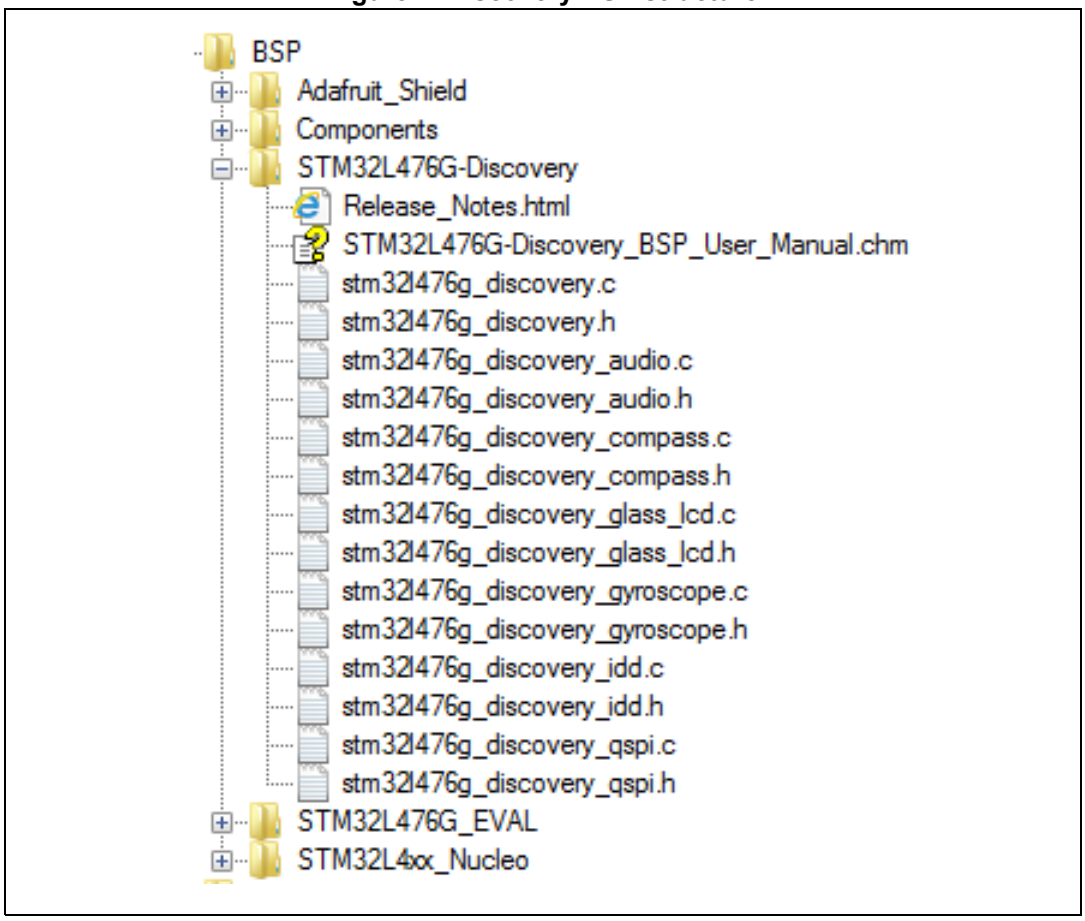
Table 1. Kernel core files

Function	Description
main.c	Main program file
stm32l4xx_it.c	Interrupt handlers for the application
k_menu.c	Kernel menu and desktop manager
k_module.c	Modules manager
k_startup.c	Demonstration startup windowing process
startup_stm32l476xx.s	Startup file

3.3 STM32L476G discovery board BSP

The board drivers available within the *stm32l476g-discoveryXXX.c/.h* files (see *Figure 7*) implement the board capabilities and the bus link mechanism for the board components (LEDs, buttons, audio, compass, glass LCD, Quad-SPI Flash memory, etc…)

Figure 7. Discovery BSP structure



The components present on the STM32L476G discovery board are controlled by dedicated BSP drivers. These are:

- The L3GD20 gyroscope in *stm32l476g_discovery_accelerometer.c/.h*
- The CS43L22 audio codec in *stm32l476g_discovery_audio.c/.h*
- The LSM303C e-Compass in *stm32l476g_discovery_compass.c/.h*
- The LCD glass 8x40 in *stm32l476g_discovery_glass_lcd.c/.h*
- The MFXSTM32L152 for Idd measurement in *stm32l476g_discovery_idd.c/.h*
- The 16 Mbytes Micron N25Q128A13 Quad-SPI Flash memory in *stm32l476g_discovery_qspi.c/.h*

4 Demonstration functional description

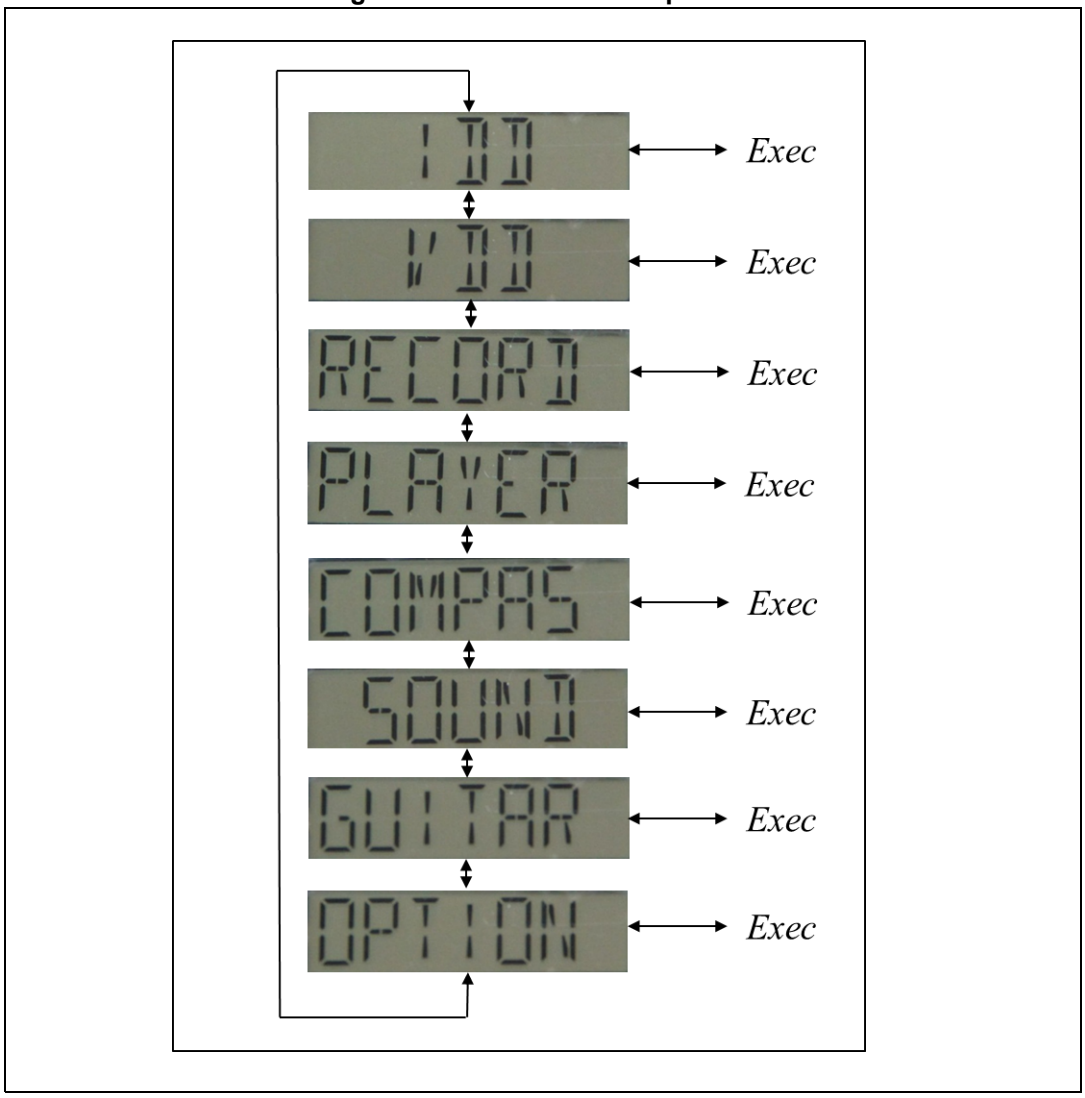
4.1 Overview

After powering the STM32L476G discovery board, the welcome message “STM32L476G-DISCOVERY DEMO” is displayed on the LCD and the first main menu application items is displayed.

4.2 Main menu

Figure 8 shows the main menu application tree with navigation possibilities:

Figure 8. Demonstration top menu



4.3 Menu navigation

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

To enter a submenu and launch *Exec function*, press the SEL pushbutton.

The SEL pushbutton designates the action of vertically pressing the top of the joystick as opposed to pressing horizontally UP, DOWN, RIGHT and LEFT.

The basic joystick key functions are defined as follows:

Table 2. Joystick key functions

Joystick Key	Function
DOWN	Go to next menu/submenu item
UP	Go to previous menu/submenu item
RIGHT / SEL	Select the demonstration application/submenu item
LEFT	Stop and exit the demonstration application/submenu item

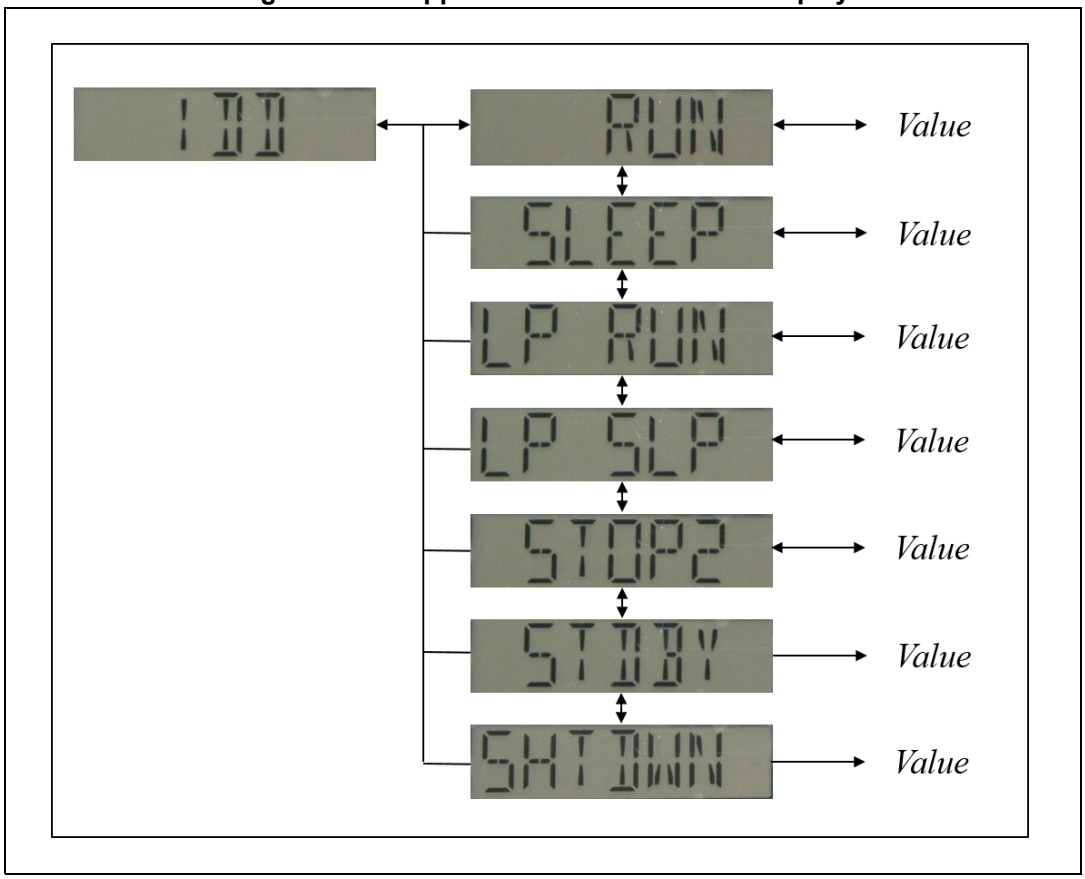
4.4 Modules

4.4.1 IDD

Overview

The IDD module application measures and displays in real time the MCU current consumption depending on the selected power mode. The current is measured and calculated using a second microcontroller from the STM32L1 Series on the board.

Figure 9. IDD application menu structure display



Value is the Idd measurement result displayed for 2.5 seconds either in milliampere (mA), microampere (µA) or nanoampere (nA).

Features

- Run mode at 24 MHz (voltage range 2), PLL Off, RTC/LSE Off, Flash ART On
- Sleep mode at 24 MHz (voltage range 2), PLL Off, RTC/LSE Off, Flash ART On
- Low-power run mode at 2 MHz, PLL Off, RTC/LSE Off, Flash ART On
- Low-power sleep mode at 2 MHz, PLL Off, RTC/LSE Off, Flash ART On
- Stop 2 mode, RTC/LSE Off, Flash ART Off
- Standby mode, RTC/LSE Off, Flash ART Off, RAM retention Off
- Shutdown mode, RTC/LSE Off, Flash ART Off

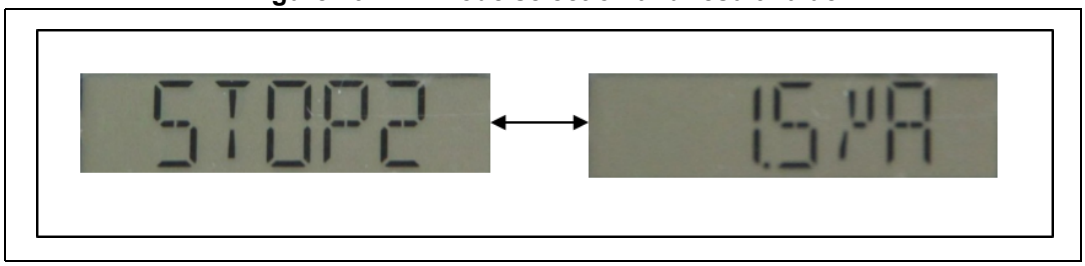
Functional description

The selection of an Idd measurement with the joystick RIGHT key executes the following sequence:

- Clear the LCD
- Enter HW components in low-power mode
- Enter MCU in low-power mode
- Wait for automatic wake up through an external event on EXTI 13. This event is the end of the Idd measurement done on the STM32L1 MCU side.
- Display the measured current value on the LCD

The measured Idd value could be for example:

Figure 10. IDD mode selection and result value

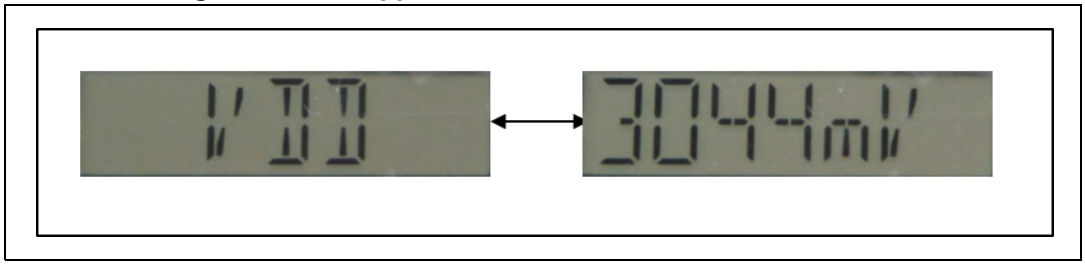


4.4.2 VDD

Overview

The VDD module measures the voltage supply with the 12-bit ADC single conversion.

Figure 11. VDD application menu selection and result value

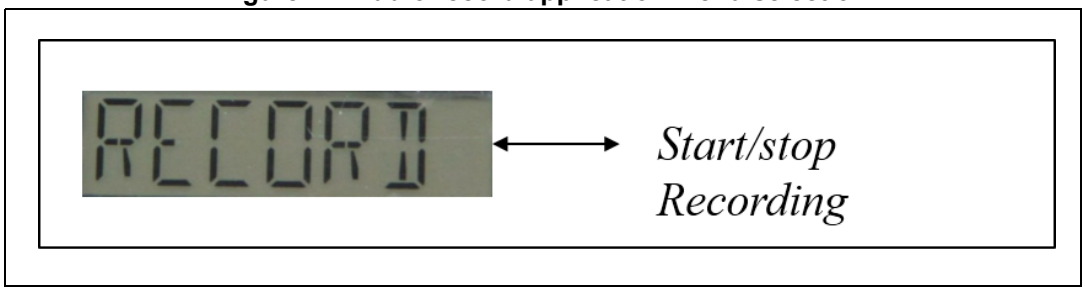


4.4.3 Audio record

Overview

The audio record module application (Record) allows to demonstrate the audio recording capability of the STM32L476G discovery board thanks to the MP34DT01 embedded digital microphone.

Figure 12. Audio record application menu selection



Features

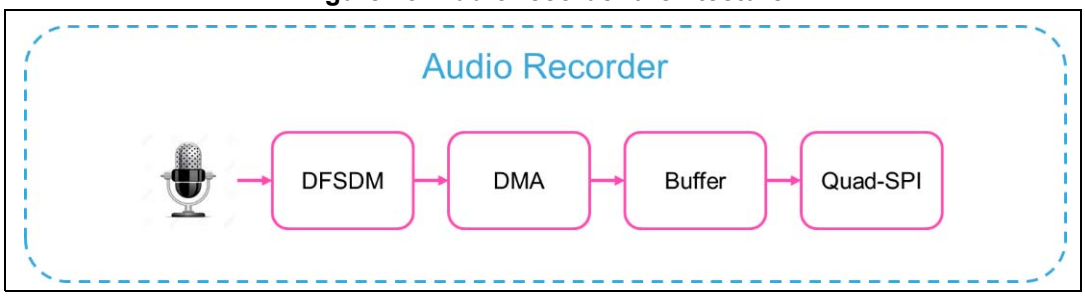
- 48 kHz audio recording in .wav format.
- Audio file stored in the Quad-SPI Flash memory (N25Q128A13)

The LED LD5 blinks continuously during the recording. Press the joystick LEFT key to stop recording and exit from the application mode. Use player application to playback the audio recording samples.

Architecture

Figure 13 shows the different audio player parts and their connections and interactions with the external components.

Figure 13. Audio recorder architecture

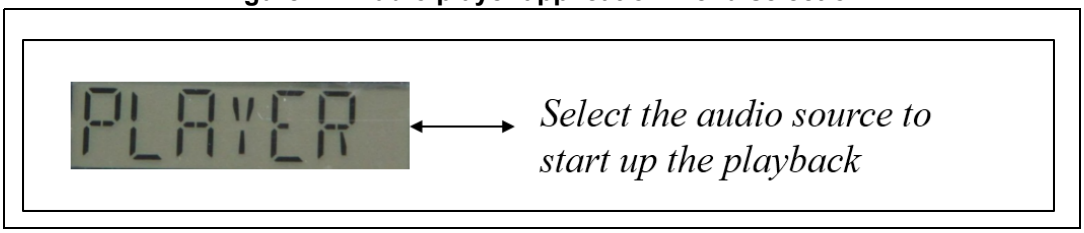


4.4.4 Audio player

Overview

The audio player module application allows to demonstrate the audio capability of the STM32L476VGT6 MCU with earphones plugged in the 3.5mm audio output jack mounted on the STM32L476G discovery board.

Figure 14. Audio player application menu selection



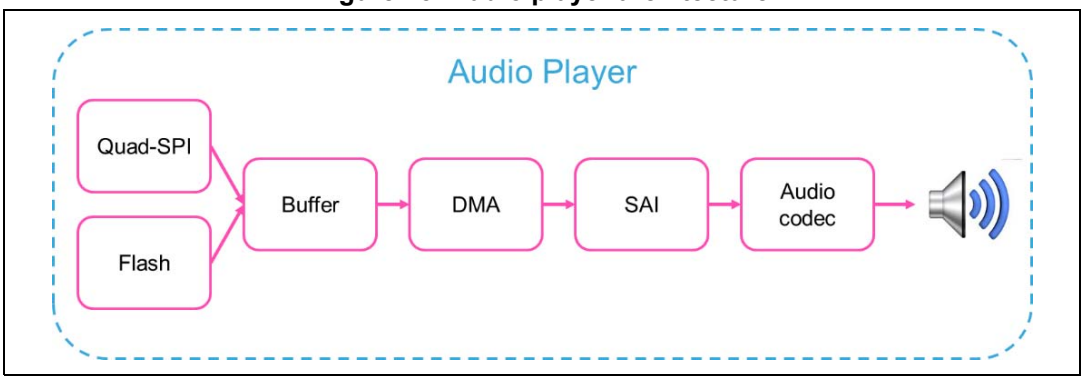
Features

- Audio is played in .wav format.
- The audio file selection is done either from the internal Flash memory or Quad-SPI Flash memory:
  - Internal Flash memory: Download file at address 0x08020000 with ST-Link Utility
  - Quad-SPI Flash memory: Audio source available after execution of the audio Record application
- Volume up/down is controlled with the joystick UP/DOWN keys.
- Playback may be paused/resumed with the joystick SEL key.

Architecture

Figure 15 shows the different audio player parts and their connections and interactions with the external components.

Figure 15. Audio player architecture

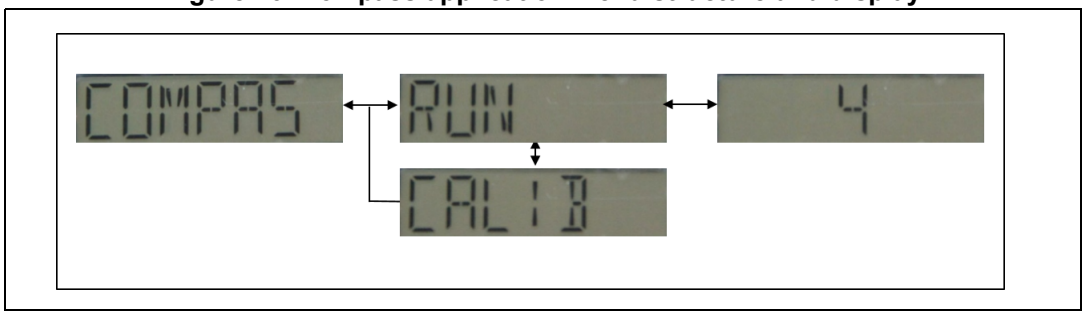


4.4.5 Compass

Overview

The Compass module application allows to demonstrate the integration of the LSM303C 3-axis electronic compass embedded on the STM32L476G discovery board.

Figure 16. Compass application menu structure and display



Features

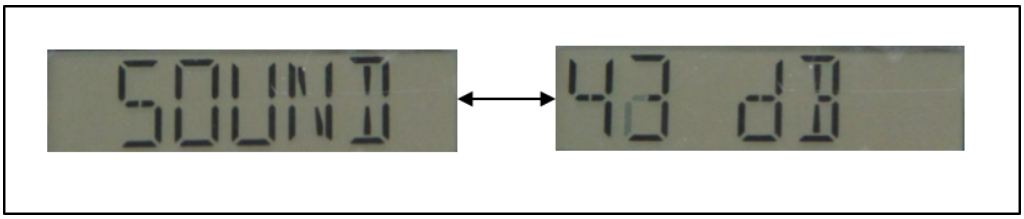
- A real time display in degree of the direction versus the magnetic north
- A calibration step is required in X/Y/Z axis to get the motion data from the application. It consists in rotating the board by 360 degrees in all 3 axis. The calibration data are saved in backup area.

4.4.6 Sound meter

Overview

The sound meter module demonstrates the perfect integration of a sound meter audio library. It uses the audio recording capability of the digital microphone embedded on the STM32L476G discovery board.

Figure 17. Sound meter application menu selection and display



Features

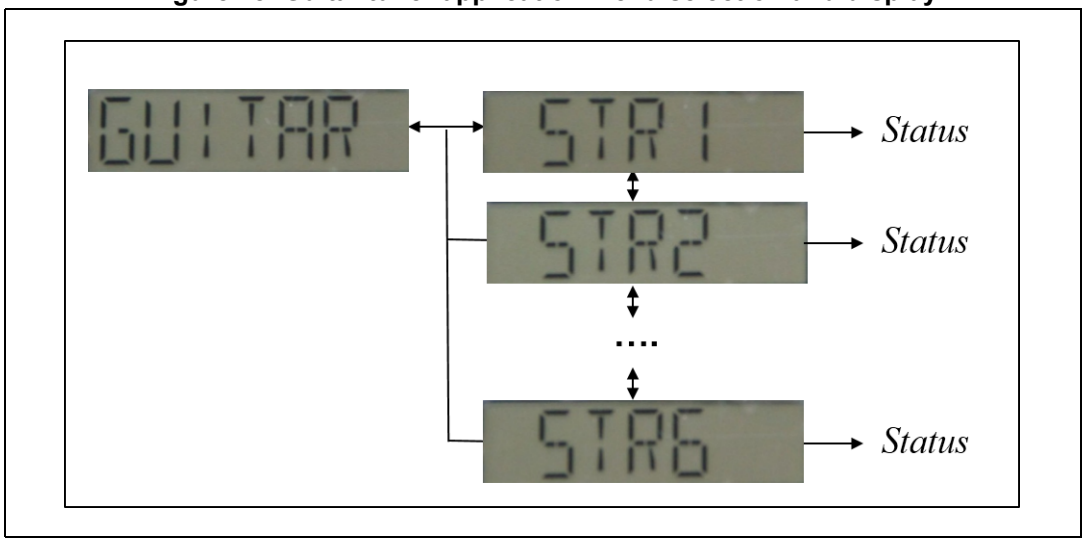
- 16-bits PCM data input with a sample rate at 48 kHz (mono)
- A-weighting pre-filter
- Real time display of the sound level every 250ms with average done every 100ms

4.4.7 Guitar tuner

Overview

The Guitar tuner module application allows to demonstrate the perfect integration of an acoustic Guitar tuner audio library. It uses the audio recording capability of the digital microphone embedded on the STM32L476G discovery board.

Figure 18. Guitar tuner application menu selection and display



Status could be any of the following display outputs:

- " ": audio sample invalid. Ensure that STM32L476G discovery board is close to the guitar
- "**OK**": string in tune
- "**++**": string needs to be tightened
- "**+**": string needs to be tightened but close to be in tune
- "**--**": string is too tightened
- "**-**": string is too tightened but close to be in tune

Features

- Standard tuning type of 6 acoustic guitar strings
- Input 16-bits PCM data with a sample rate at 8 kHz (mono)
- Precision lower than 0.5 Hz
- Real time display every 256ms

The standard tuning method is based on the following expected frequencies per string:

Table 3. Frequencies per string

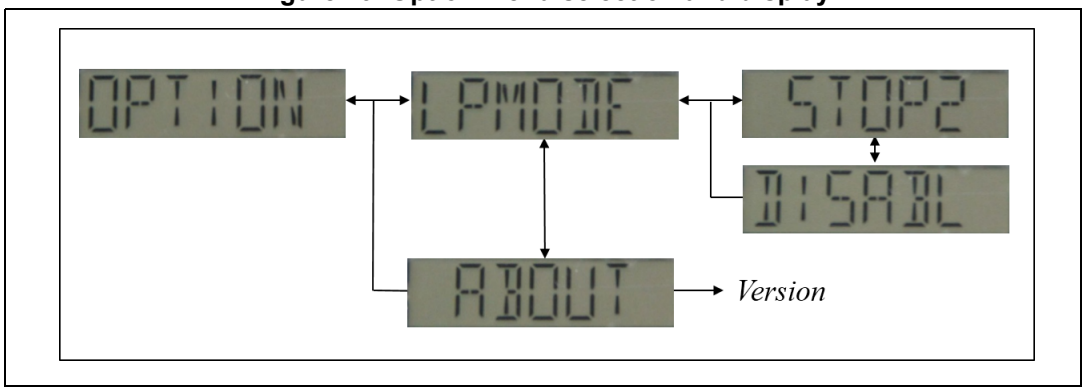
String 1 (Hz) (the thickest string)	String 2 (Hz)	String 3 (Hz)	String 4 (Hz)	String 5 (Hz)	String 6 (Hz) (the thinnest string)
E1 = 82.41	A1 = 110	D2 = 146.83	G2 = 196	B2 = 246.94	E3 = 329.63

4.4.8 Option

Overview

The option menu level provides the system control and information about the demonstration firmware.

Figure 19. Option menu selection and display



Features

- Low-power mode selection
  - Stop2: the MCU enters automatically into Stop2 mode after X seconds of inactivity
  - Disable: the MCU never enters in low-power mode (except in IDD demonstration)

    The disable mode is available when running in USB-powered mode and not in battery mode to not empty the battery.
- Demonstration version (about)
  - Display STM32L476G discovery firmware version and copyright

4.5 Audio player control

Within the audio player application, the joystick key functions are enhanced and are described in the following *Table 4*:

Table 4. Audio player control joystick key functions

Joystick key	Function	Brief description
RIGHT	Play	Changes the audio player state to “AUDIOPLAYER_PLAY”. Reads the wave file from the storage Flash memory. Sets the frequency. Starts or resumes the audio task. Starts playing audio stream from a data buffer using “BSP_AUDIO_OUT_Play” function in BSP audio driver.
SEL	Pause / Resume	Suspends the audio task when playing and pauses the audio file stream. Resumes the audio task and the audio file stream when suspended.

Table 4. Audio player control joystick key functions (continued)

Joystick key	Function	Brief description
LEFT	Stop & Exit	Closes the wave file from storage flash memory. Suspends the audio task. Stops audio playing. Changes the audio player state to “AUDIOPLAYER_STOP”.
UP	Volume Up	Increases the volume.
DOWN	Volume Down	Decreases the volume.

5 Demonstration firmware settings

5.1 Clock control

The demonstration firmware takes benefit of the STM32L476VGT6 clock tree by selecting dynamically the clock configuration required by the running application/module when running in low-power mode.

The following clock configurations are used in the demonstration firmware:

Table 5. Clock configurations

Clock configuration	Application/module USB-powered	Application/module battery
SYSCLK: 2MHz MSI (LPRUN voltage range 2)	-	Demonstration startup IDD application
SYSCLK: 16MHz PLL from MSI 16MHz (RUN voltage range 2)	-	Sound Meter Compass
SYSCLK: 80MHz (PLL) from MSI 8MHz (RUN voltage range 1)	Demonstration startup All applications	-

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

Table 6. Oscillators and PLL description

Oscillators	Application/module USB-powered	Application/module battery
MSI from 2MHz to 16MHz	Demonstration startup	Demonstration startup IDD application Sound meter Compass
LSE	LCD glass display	LCD glass display
LSI	Internal Watchdog for automatic FW reset (error management)	-
PLL main	Demonstration startup All applications	Sound meter Compass
PLLSAI1 PLLSAI1_VCO= 8 MHz * PLLSAI1N = 8 * 43 = VCO_344M SAI_CK_x = PLLSAI1_VCO/PLLSAI1P = 344/7 = 49.142 MHz	Player Record Sound meter Guitar tuner	Sound meter

5.2 Peripherals

The following peripherals are used in the demonstration firmware:

Table 7. Used peripherals

Used peripherals	Application/module
ADC	VDD application
CRC	Audio applications
DFSDM	Audio record applications
DMA	Audio player and record applications
EXTI	Menu navigation + joystick + low-power mode + audio applications
FLASH	Storage in Audio player application System settings Low-power mode application
GPIO	All applications
I2C	Low-power mode application
IWDG	Internal Watchdog (FW reset in case of error)
LCD	LCD glass display
PWR	Low-power mode application
RCC	All applications
QSPI	Audio player and record applications
SAI	Audio applications
SPI	Compass application

5.3 Interrupts / wakeup pins

The following interrupts are used in the demonstration firmware:

Table 8. Demonstration firmware interrupts

Interrupts	Application/module	Priority, SubPriority (highest=0, 0)
DMA1 Channel1	Audio applications (SAI1)	5, 0
DMA1 Channel4	Audio applications (DFSDM0)	6, 0
EXTI Line 0	Joystick SEL key	15, 0
EXTI Line 1	Joystick LEFT key	15, 0
EXTI Line 2	Joystick RIGHT key	15, 0
EXTI Line 3	Joystick UP key	15, 0
EXTI Line 5	Joystick DOWN key	15, 0

Table 8. Demonstration firmware interrupts (continued)

Interrupts	Application/module	Priority, SubPriority (highest=0, 0)
EXTI Line 13	IDD application (Interrupt from MFXSTM32L152 MCU)	15, 15
SysTick	CortexM4 System Timer for OS Tick	15, 0

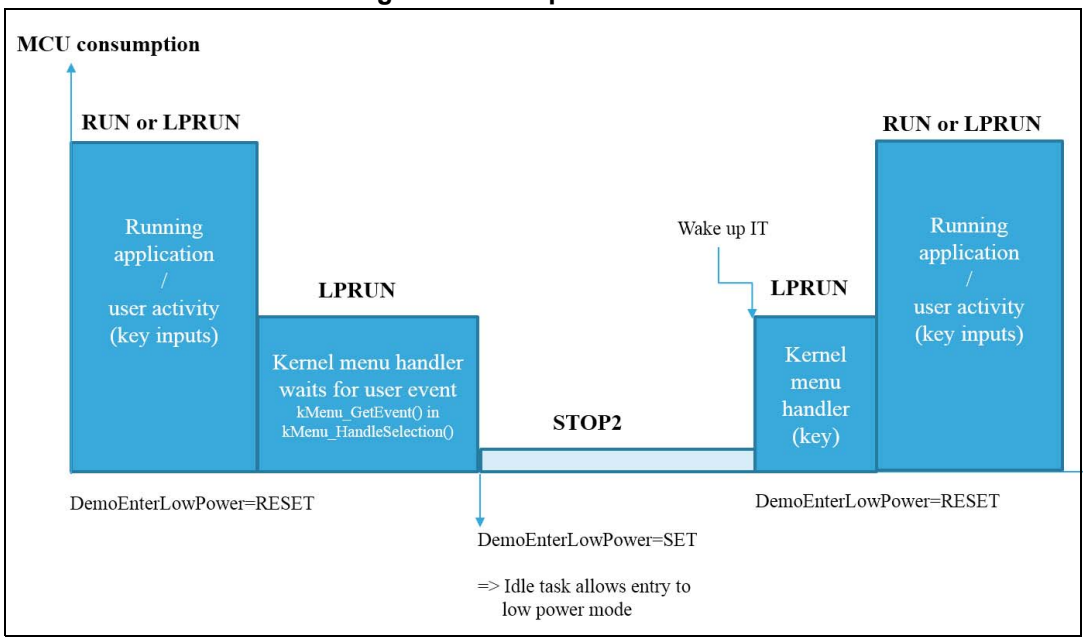
The wakeup pin 2 is used to wake up the MCU from Standby and Shutdown modes.

5.4 Low-power strategy

The STM32CubeL4 demonstration firmware is designed to highlight the low-power consumption capabilities of both the STM32L476VG MCU and the STM32L476G discovery board.

The *Figure 20* illustrates the low-power strategy put in place in the STM32Cube demonstration firmware based on no running application and user activity monitoring.

Figure 20. Low-power scheme



5.5 FreeRTOS resources

The STM32L476G demonstration firmware is designed on top of CMSIS-OS drivers based on FreeRTOS. The resources used in the firmware demonstration are listed hereafter.

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

5.5.1 Tasks

Table 9. Task description

Task entry point	Description	Function (File)	Stack size (words)	Priority
StartThread	Main application core	Main.c	2 * 128	osPriorityNormal
AudioPlayer_Thread	Audio player application	AudioPlayer_Init() (Audioplayer.c)	4 * 128	osPriorityHigh
AudioRecorder_Thread	Audio recorder application	AudioRecorder_Init() (Audiorecorder.c)	4 * 128	osPriorityHigh
Idle task (FreeRTOS)	Implement Idle hook application vApplicationIdleHook() used to enter low-power mode	vApplicationIdleHook() (Main.c)	128	osPriorityHigh

Stack size: 128 is the value defined for configMINIMAL_STACK_SIZE in *FreeRTOSConfig.h*

5.5.2 Message queues

Table 10. Message queues

QueueId	Description	Function (File)	Queue depth (word)
JoyEvent	Queue to receive input key event	kMenu_Init() (K_menu.c)	1
AudioEvent	Audio player input event	AudioPlayer_Init() (Audioplayer.c)	1
osMsdId part of AudioRecorder_ContextTypeDef structure	Audio Recorder input event	AudioRecorder_Init() (Audiorecorder.c)	1

5.5.3 Mutex

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

The mutex *DemoLowPowerMutex* is released upon wake-up from an EXTI lines associated to joystick buttons.

5.5.4 Heap

The Heap size is defined in FreeRTOSConfig.h as follows:

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

Heap usage in the firmware demonstration is dedicated to:

- OS resources (tasks, queues, mutexes, memory allocation)
- Application memory allocation requirements

Table 11. Heap usage

Applications	Description	Function (File)	Memory requirements (bytes)
Audio Record	Record buffer	AudioRecorder_Start() (Audiorecorder.c)	1024
Guitar tuner	Record buffer	GuitarTuner_Run() (Guitartuner.c)	4096
Guitar tuner Guitar tuner library	Processing buffer used by Guitar tuner library	GuitarTuner_Init() (Guitartuner.c)	< 16 Kbytes
Soundmeter	Record buffer	SoundMeter_Run() (Soundmeter.c)	960
Soundmeter Soundmeter library	Processing buffer used by Soundmeter library	SoundMeter_Init() (Soundmeter.c)	< 6 Kbytes

The demonstration firmware implements a hook in *main.c* to control the memory allocation in the heap

```
1195 /**
1196   * @brief  Application Malloc failure Hook
1197   * @param  None
1198   * @retval None
1199   */
1200 void vApplicationMallocFailedHook(void)
1201 {
1202   Error_Handler();
1203 }
```

5.6 Programming firmware application

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

There are two ways of programming the STM32L476G discovery board.

5.6.1 Using binary file

Upload the binary STM32CubeDemo_STM32L476G-Discovery-VX.Y.Z.hex from the firmware package available under Projects\STM32L476G-Discovery\Demonstrations\Binary using your preferred in-system programming tool.

5.6.2 Using preconfigured projects

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

- Open the application folder: Projects\STM32L476G-Discovery \Demonstrations\
- Chose the desired IDE project (EWARM for IAR, MDK-ARM for Keil)
- Double click on the project file (for example Project.eww for EWARM)
- Rebuild all files: Go to Project and select Rebuild all
- Load the project image: Go to Project and select Debug
- Run the program: Go to Debug and select Go

The demonstration software as well as other software examples that allow the user to discover the STM32 microcontroller features are available on ST website at www.st.com/stm32l4-discovery.

6 Demonstration firmware footprints

This section provides the memory requirements for all the demonstration modules. The aim is to have an estimation of memory requirement in case of suppression or addition of a module or feature.

The footprint information is provided for the following environment:

- Tool chain: IAR 7.40.1
- Optimization: high size
- Board: STM32L476G discovery.

Table 12 shows the code memory, data memory and the constant memory used for each application module file and related libraries.

Table 12. Modules footprint

Module	Code [Byte]	Data [byte]	Const [byte]
IDD	832	237	264
VDD	324	142	68
RECORD	1112	-	68
PLAYER	1212	-	68
COMPAS	1052	108	124
SOUND + SoundMeter library	638 4338	3932 12	508 8
GUITAR + GuitarTuner library	592 6704	12416 -	236 248
Main application (Top menu + Option submenu)	274	276	376

7 Kernel description

7.1 Overview

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

- Hardware and modules initialization:
  - BSP initialization (LEDs, joystick keys, LCD, audio, Quad-SPI and compass)
- Main menu management.
- Memory management
- Storage management (Quad-SPI Flash memory)
- System settings

7.2 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 during the middleware components:

- LEDs and buttons
- LCD
- Backup area
- External Quad-SPI Flash memory
- Audio codecs
- MEMS

Upon full initialization phase, the kernel adds and links the system and user modules to the demonstration core waiting to execute a menu entry point.

7.3 Kernel processes and tasks

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

- Start thread: this task initializes the OS resources required by the demonstration framework and then starts the demonstration.

```
/**
  * @brief  Start task
  * @param  argument: pointer that is passed to the thread function as start argument.
  * @retval None
  */
static void StartThread(void const * argument)
{
  osMutexDef_t mutex_lowpower;

  /* Create mutex to handle low power mode */
  DemoLowPowerMutex = osRecursiveMutexCreate(&mutex_lowpower);

  /* Start Demo */
  kDemo_Start();
}
```

7.4 Kernel event manager

The kernel provides services to manage events mainly here user key input (*k_menu.h* file)

```
void kMenu_SendEvent(uint16_t JOY_Pin);
JOYState_TypeDef kMenu_GetEvent(uint32_t Delay);
```

kMenu_SendEvent() is called from the interrupt IRQ callback (HAL_GPIO_EXTI_Callback()) to send the event message to the Kernel event mailbox.

kMenu_GetEvent() is called by either the kernel event manager in kMenu_HandleSelection() or directly by the running module application expecting user input keys.

7.5 Module manager

The main demonstration menu is initialized and launched by the Start thread.

The modules are managed by the kernel; this later is responsible of initializing the modules, initializing hardware resources relative to the modules and initializing the system menu.

Each module should provide the following functionalities and properties:

1. Graphical component structure.
2. Method to startup the module.
3. Method to manage low-power mode (optional)
4. The application task
5. The module background process (optional)
6. Specific configuration
7. Error management

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

```
260  void kModule_Init(void)
261  {
262    kModule_Add(MODULE_MAIN_APP, ModuleAppMain);
263    kModule_Add(MODULE_IDDMEASURE, ModuleIddMeasure);
264    kModule_Add(MODULE_BATTERYMEASURE, ModuleBatteryMeasure);
265    kModule_Add(MODULE_COMPASS, ModuleCompass);
266    kModule_Add(MODULE_SOUNDMETER, ModuleSoundMeter);
267
268    if(PowerSupplyMode != SUPPLY_MODE_BATTERY)
269    {
270      kModule_Add(MODULE_AUDIOPLAYER, ModuleAudioPlayer);
271      kModule_Add(MODULE_AUDIORECORDER, ModuleAudioRecorder);
272      kModule_Add(MODULE_GUITARTUNER, ModuleGuitarTuner);
273    }
274  }
```

A module is a set of function and data structures that are defined in a data structure that provides all the information and pointers to specific methods and functions to the kernel. 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 ID. When two modules have the same ID, the Kernel rejects the second one. The module structure is defined as follows:

```
56  typedef struct
57  {
58    uint8_t         kModuleId;
59    KMODULE_RETURN (*kModulePreExec)(void);
60    KMODULE_RETURN (*kModuleExec)(void);
61    KMODULE_RETURN (*kModulePostExec)(void);
62    KMODULE_RETURN (*kModuleResourceCheck)(void);
63  } K_ModuleItem_Typedef;
```

- Id: unique module identifier
- Name: pointer to the module name
- kModuleId: module identifier
- kModulePreExec: pre-execution function to allocate resources prior to the execution
- kModuleExec: execution function
- kModulePostExec: post-execution function to release resources
- kModuleResourceCheck: function to check the resource requirements after the module was added to the kernel

7.6 Backup and settings configuration

The STM32CubeL4 demonstration firmware saves the kernel and module settings in the bank2 of the internal Flash memory (address 0x08080000).

```
typedef struct
{
  /* IDD application */
  IddBackupData_TypeDef      idd;
  /* Compass application */
  CompassBackupData_TypeDef  compass;
  /* Global settings */
  SettingsBackupData_TypeDef settings;
} DemoBackupData_TypeDef;
```

```
/* Exported constants --------------------------------------------------------*/
#define DEMOBACKUP_AREA_ADDRESS 0x08080000   /* Backup data in first section of Bank2 */

```

```
typedef enum
{
  BACKUP_IDD        = 0x00,  /* IDD application backup data     */
  BACKUP_COMPASS    = 0x01,  /* Compass application backup data */
  BACKUP_SETTINGS   = 0x02   /* Kernel settings backup data     */
} DemoBackupId;
```

The following APIs allow to save or restore it from the backup area.

```
uint32_t  SystemBackupRead(DemoBackupId Id, void *Data);
void      SystemBackupWrite(DemoBackupId Id, void *Data);
```

7.7 Adding a new module

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

1. Define unique ID of the new module in *k_config.h* file.
2. k_ModuleAdd() function should be called in k_ModuleInit() from KDemo_Initialization() with unique ID defined in step1.
3. Modify main module in *main_app.c* file to add the call to this new module in ConstMainMenuItems[] table.

```
/* Main Menu */
const tMenuItem ConstMainMenuItems[] =
{
    {"  IDD "       ,0,0,   SEL_MODULE,  MODULE_IDDMEASURE    , NULL, NULL, NULL, NULL},
    {"  VDD "       ,0,0,   SEL_MODULE,  MODULE_BATTERYMEASURE, NULL, NULL, NULL, NULL},
    {"RECORD"       ,0,0,   SEL_MODULE,  MODULE_AUDIORECORDER , NULL, NULL, NULL, NULL},
    {"PLAYER"       ,0,0,   SEL_MODULE,  MODULE_AUDIOPLAYER   , NULL, NULL, NULL, NULL},
    {"COMPAS"       ,0,0,   SEL_MODULE,  MODULE_COMPASS       , NULL, NULL, NULL, NULL},
    {" SOUND"       ,0,0,   SEL_MODULE,  MODULE_SOUNDMETER    , NULL, NULL, NULL, NULL},
    {"GUITAR"       ,0,0,   SEL_MODULE,  MODULE_GUITARTUNER   , NULL, NULL, NULL, NULL},
    {"OPTION"       ,0,0,   SEL_SUBMENU, MODULE_NONE          , NULL, NULL, (const tMenu*)&OptionMenu, NULL },
    {""             ,0,0,   SEL_NOTHING, MODULE_NONE          , NULL, NULL, NULL, NULL}
};
```

8 Revision history

Table 13. Document revision history

Date	Revision	Changes
17-Jul-2015	1	Initial release.