STM32F4-Discovery
(Le firmware)

Table des matières

Table des figures

1 Introduction

L'initiative STMCube a été lancée par STMicroelectronics pour faciliter la vie des développeurs en réduisant les efforts de développement, le temps et le coût. STM32Cube couvre l'éventail de produits STM32.

STM32Cube Version 1.x comprend:

- le STM32CubeMX, un outil de configuration graphique qui permet de générer du code C d'initialisation à l'aide d'assistants graphiques,
- une plateforme de logiciel embarqué complète, fournie par série (telle que STM32CubeF4 pour la série STM32F4) et offrant:
    - le STM32Cube HAL assurant une portabilité optimale à travers l'éventail STM32,
    - un ensemble cohérent de composants middleware tels que RTOS, USB, TCP/IP et graphisme,
    - tous les utilitaires logiciels intégrés sont livrés avec un ensemble complet d'exemples.

Ce manuel d'utilisation décrit comment démarrer avec le progiciel STM32CubeF4. La section 1 décrit les principales caractéristiques du firmware STM32CubeF4, qui fait partie de l'initiative STMCube.

La Section 2 et la Section 3 fournissent une vue d'ensemble de l'architecture STM32CubeF4 et de la structure du progiciel.

2 Principales caractéristiques du STM32CubeF4

STM32CubeF4 regroupe, dans un seul paquet, tous les composants logiciels génériques embarqués requis pour développer une application sur les microcontrôleurs STM32F4. Conformément à l'initiative STMCube TM, cet ensemble de composants est très portable, non seulement dans la série STM32F4 mais aussi dans d'autres séries STM32. STM32CubeF4 est entièrement compatible avec le générateur de code STM32CubeMX qui permet à l'utilisateur de générer le code d'initialisation. Le package comprend une couche d'abstraction matérielle de bas niveau (HAL) qui couvre le matériel du microcontrôleur, ainsi qu'un vaste ensemble d'exemples fonctionnant sur les cartes STMicroelectronics.

Le package STM32CubeF4 contient également un ensemble de composants middleware avec les exemples correspondants:

- Un hôte USB complet et une pile de périphériques prenant en charge de nombreuses classes:
    - classes d'hôtes: HID, MSC, CDC, Audio, MTP
    - classes de dispositifs: HID, MSC, CDC, Audio, DFU
- Graphisme:
    - STemWin, une solution de pile graphique professionnelle disponible en format binaire et basée sur la solution emWin de SEGGER, un partenaire de ST,
    - LibJPEG, une implémentation open source sur STM32 pour l'encodage et le décodage des images JPEG,
- Mise en œuvre de CMSIS-RTOS avec la solution open source FreeRTOS,
- Système de fichiers FAT basé sur une solution FatFS open source,
- pile TCP/IP basée sur une solution open source LwIP,
- Couche sécurisée SSL / TLS basée sur le logiciel libre PolarSSL.

Une démonstration mettant en œuvre tous ces composants middleware est également fournie dans le package STM32CubeF4.

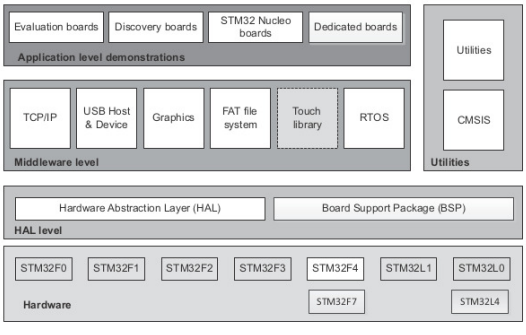

Figure 1 Composant du firmware STM32CubeF4.

3 Présentation de l'architecture du STM32CubeF4

Le firmware STM32CubeF4 est construit autour de trois niveaux indépendants qui peuvent facilement interagir les uns avec les autres comme décrit dans la figure 2↓:

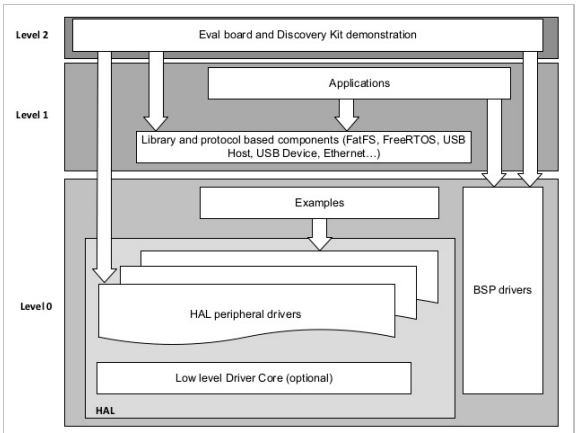

Figure 2 Architecture du firmware STM32CubeF4.

3.1 Le niveau 0

Ce niveau est divisée en trois sous-couches:

1. *Le BSP (Board Support Package)*
   Cette couche offre un ensemble d'APIs liées aux composants matériels sur les cartes matérielles (codec audio, amplificateur d'E/S, écran tactile, pilote SRAM, pilotes LCD et autres). Cette couche est composée de deux parties:

   1. *le composant*: c'est le pilote qui est lié au périphérique externe sur la carte et n'est pas lié au STM32. Le pilote de composant fournit des API spécifiques aux composants externes du pilote BSP et peut être porté à n'importe quelle carte,
   2. *le pilote BSP*: il permet de lier le pilote de composant à une carte spécifique et fournit un ensemble d'APIs conviviales. La règle d'appellation de l'API est `BSP_FUNCT_Action ()`: ex. `BSP_LED_Init ()`, `BSP_LED_On ()`

   Le BSP est basé sur une architecture modulaire qui lui permet d'être facilement porté à n'importe quel matériel par simple implémentation des routines de bas niveau.

2. *Le HAL (Hardware Abstraction Layer)*
   Cette couche fournit les pilotes de bas niveau et les méthodes d'interfaçage du matériel pour interagir avec les couches supérieures (application, bibliothèques et piles). Il fournit des APIs génériques, multi-instances et orientées fonction, qui permettent de décharger l'implémentation de l'application utilisateur en fournissant des processus prêts à l'emploi. Par exemple, pour les périphériques de communication (I2S, UART ...), il fournit des API permettant:

   - d'initialiser et de configurer le périphérique,
   - de gérer le transfert de données basé sur le processus
     - d'interrogation (polling),
     - d'interruption,
     - ou de DMA,
   - de gérer les erreurs de communication pouvant survenir pendant la communication.

   Les API des pilotes HAL sont divisées en deux catégories:

   - les API génériques qui fournissent des fonctions communes et génériques à toutes les API de la série STM32, et
   - les APIs d'extension qui fournissent des fonctions spécifiques et personnalisées pour une famille ou un numéro spécifique.

3. *Exemples d'utilisation des périphériques de base*
   Cette couche contient des exemples de fonctionnement de base des périphériques STM32F4 uniquement à l'aide des ressources HAL et BSP.

3.2 Le niveau 1

Ce niveau est divisée en deux sous-couches:

1. *Composants middleware*
   Un ensemble de bibliothèques couvrant les bibliothèques d'hôtes et de périphériques USB, STemWin, LibJPEG, FreeRTOS, FatFS, LwIP et PolarSSL. Les interactions horizontales entre les composants de cette couche sont effectuées directement en appelant les API de fonctionnalités tandis que l'interaction verticale avec les pilotes de bas niveau est effectuée via des callbacks spécifiques et des macros statiques implémentées dans l'interface d'appel de système de bibliothèque. Par exemple, le FatFs implémente le pilote d'E / S de disque pour accéder au lecteur de microSD TM ou à la classe de stockage de masse USB.
   Les principales caractéristiques de chaque composant middleware sont les suivantes:

   1. *Bibliothèques d'hôtes USB*

      - Plusieurs classes d'USB sont prises en charge (Stockage de masse, HID, CDC, DFU, AUDIO, MTP).
      - Prise en charge des fonctionnalités de transfert multi-paquets: possibilité d'envoyer de grandes quantités de données sans les diviser en paquets de tailles maximales.
      - Utilisation de fichiers de configuration pour modifier le noyau et la configuration de la bibliothèque sans modifier le code de la bibliothèque (en lecture seule).
      - Utilisation de structures de données alignées sur 32 bits pour gérer le transfert basé sur le DMA dans les modes vitesse élevée.
      - Prend en charge les instances multi-core USB OTG à partir du niveau l'utilisateur via le fichier de configuration (permet le fonctionnement avec plus d'un hôte USB/équipement périphérique).
      - RTOS et fonctionnement autonome.
      - Le lien avec le pilote de bas niveau s'effectue via une couche d'abstraction à l'aide du fichier de configuration pour éviter toute dépendance entre la bibliothèque et les pilotes de bas niveau.

   2. *pile graphique STemWin*

      - Solution de niveau professionnel pour le développement de l'interface utilisateur basée sur la solution emWin de SEGGER.
      - Pilotes d'affichage optimisés.
      - Outils logiciels pour la génération de code et l'édition bitmap (STemWin Builder ...).

   3. *LibJPEG*

      - Norme Open Source.
      - Implémentation en C du codage et décodage d'images JPEG.

   4. *FreeRTOS*

      - Norme Open Source.
      - Couche de compatibilité CMSIS.
      - Fonctionnement sans tick (*Tickless operation*) en mode faible consommation.
      - Intégration avec tous les modules middleware STM32Cube.

   5. *Système de fichiers FAT*

      - Bibliothèque FAT open source FATFS.
      - Support de nom de fichier long.
      - Support multi-lecteur dynamique.
      - RTOS et fonctionnement autonome.
      - Exemples avec classe de stockage de masse hôte microSD et USB.

   6. *Pile TCP/IP LwIP*

      - Norme Open Source.

- RTOS et fonctionnement autonome.

2. *Exemples basés sur les composants middleware*
   Chaque composant middleware est livré avec un ou plusieurs exemples (appelés aussi Applications) montrant comment l'utiliser. Des exemples d'intégration qui utilisent plusieurs composants middleware sont également fournis.

3.3 Le niveau 2

Ce niveau est composé d'une couche unique qui est une démonstration graphique globale et en temps réel, basée sur:

- la couche de service middleware,
- la couche d'abstraction de bas niveau, et
- les applications qui utilisent les périphériques de base pour des fonctions basées sur la carte.

4 Vue d'ensemble du progiciel STM32CubeF4

4.1 Périphériques et matériel STM32F4 pris en charge

STM32Cube offre une couche d'abstraction matérielle (HAL) très portable, construite autour d'une architecture modulaire et générique permettant aux couches supérieures, middleware et application d'exécuter ses fonctions sans une connaissance approfondie de la MCU utilisée. Cela améliore la réutilisation du code de la bibliothèque et garantit une portabilité facile d'un dispositif à l'autre.

Le STM32CubeF4 offre un support complet pour toutes les MCU de la série STM32F4. L'utilisateur n'a qu'à définir la bonne macro dans `stm32f4xx.h`.

Dans le cas de la carte STM32F4-Discovery dont le MCU est STM32F407VG, il faudra définir la macro `STM32F407xx` qui en est la même pour tous les MCU STM32F407VG, STM32F407VE, STM32F407ZG, STM32F407ZE, STM32F407IG et STM32F407IE.

STM32CubeF4 dispose d'un riche ensemble d'exemples et de démonstrations à tous les niveaux, ce qui facilite la compréhension et l'utilisation des pilotes HAL et/ou des composants middleware.

STM32F4 prend en charge les cartes Nucleo-64 et Nucleo-144. Ces cartes supportent l'écran LCD Adafruit, les écrans Arduino UNO qui intègrent un connecteur microSD, et un joystick en plus de l'écran LCD.

Les pilotes de shields Arduino sont fournis dans le composant BSP. Leur utilisation est illustrée par un firmware de démonstration.

Le firmware STM32CubeF4 peut fonctionner sur tout matériel compatible. Si la carte de l'utilisateur a les mêmes caractéristiques matérielles que la carte ST (LED, écran LCD, boutons-poussoirs et autres), l'utilisateur a juste à mettre à jour les pilotes BSP pour porter les exemples fournis sur sa carte.

4.2 Présentation du package de microprogramme

Le firmware STM32CubeF4 est fourni en un seul fichier zip avec la structure illustrée dans la figure 3↓.

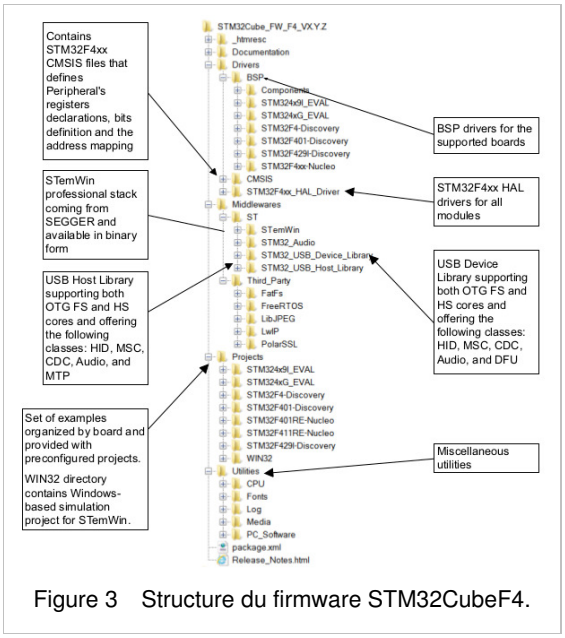

Figure 3 Structure du firmware STM32CubeF4.

Pour chaque carte, un ensemble d'exemples est fourni avec des projets préconfigurés pour les chaînes d'outils EWARM, MDK-ARM,

TrueSTUDIO et SW4STM32.

La Figure 4↓ montre la structure du projet pour la carte STM324xG-EVAL. La structure est identique pour les autres cartes.

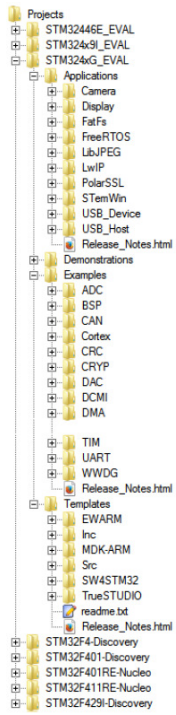

Figure 4 Les exemples fournis par le firmware STM32CubeF4.

Les exemples sont classés en fonction du niveau du STM32Cube auquel ils s'appliquent, et sont nommés comme suit:

- Les exemples au niveau 0 sont appelés `Examples` et utilisent des pilotes HAL sans composant middleware,
- Les exemples du niveau 1 sont appelés `Applications` et fournissent des cas d'utilisation typiques de chaque composant middleware,
- Les exemples du niveau 2 sont appelés `Demonstration` et implémentent les composants HAL, BSP et middleware.

Un projet modèle est fourni pour permettre aux utilisateurs de construire rapidement toute application sur une carte donnée.
Tous les exemples ont la même structure:

- le répertoire `/inc` contient tous les fichiers d'en-têtes,
- le répertoire `/src` pour le code source,
- les répertoires `/EWARM`, `/MDK-ARM`, `/TrueSTUDIO` et `/SW4STM32` contiennent le projet pré-configuré pour chaque chaîne d'outils,
- le fichier `readme.txt` décrit le comportement de l'exemple et l'environnement requis pour le faire fonctionner.

Dans le cas du STM32F4-Discovery, le firmware (`STM32F4-Discovery_FW_V1.1.0`) inclut 27 exemples, 3 application et 1 démonstration.

5 Guide de démarrage

5.1 Exécuter un premier exemple

Cette section explique comment il est simple d'exécuter un premier exemple avec le STM32CubeF4. À titre d'exemple, considérons un simple exemple d'allumage d'une LED sur la carte STM32F4-Discovery:

1. Après avoir téléchargé le progiciel STM32CubeF4, décompressez-le dans un répertoire sélectionné.
2. Accéder à `/Projects/STM32F4-Discovery/Examples`.

3. Ouvrir le répertoire `/GPIO` puis `/GPIO_EXTI`.

4. Ouvrez le projet avec la chaîne d'outils préférée.

5. Reconstruire tous (*Rebuild all*) les fichiers et charger l'image dans la mémoire de la cible.

6. Exécute l'exemple: chaque fois qu'on appuie sur le bouton utilisateur, la LED bascule (pour plus de détails, voir le fichier `readme.txt` de l'exemple).

Un aperçu rapide de la façon d'ouvrir, de construire et d'exécuter un exemple avec la chaîne d'outils SW4STM32 est donné ci-après:

1. Ouvrir la chaîne d'outils SW4STM32.

2. Cliquer sur `File -> Switch Workspace -> Other` et accédez au répertoire de l'espace de travail SW4STM32.

3. Cliquer sur `File -> Import`, sélectionner `General -> Existing Projects into Workspace`, puis cliquez sur `Next`.

4. Accéder au répertoire de l'espace de travail SW4STM32, sélectionnez le projet.

5. Reconstruire tous les fichiers du projet: sélectionner le projet dans la fenêtre *Project explorer* puis cliquer sur Project -> build project.

6. Exécuter le programme: `Run -> Debug (F11)`.

5.2 Comment développer une application

Cette section décrit les étapes requises pour créer une application à l'aide de STM32CubeF4.

5.2.1 Créer un projet

Pour créer un nouveau projet à partir du modèle de projet fourni pour la carte STM32F4-Discovery dans le répertoire `/Projects/STM32F4-Discovery/Templates` ou de tout projet disponible sous `/Projects/STM32F4-Discovery/Examples` ou `/Projects/STM32F4-Discovery/Applications`.

Le modèle fournit une fonction main contenant une boucle vide, c'est un bon point de départ pour permettre à l'utilisateur de se familiariser avec les définitions des paramètres de projet pour le STM32CubeF4. Il présente les caractéristiques suivantes:

1. Il contient des sources des pilotes HAL, CMSIS et BSP qui sont les composants minimum requis pour développer du code pour une carte.

2. Il contient les chemins d'inclusion de tous les composants du firmware.

3. Il définit le dispositif STM32F4 pris en charge, permettant d'avoir la bonne configuration pour les pilotes CMSIS et HAL.

4. Il fournit des fichiers utilisateur prêts à l'emploi, pré-configurés comme suit:

    - HAL est initialisé,
    - ISR SysTick est implémenté pour les besoins de `HAL_Delay()`,
    - L'horloge système est configurée avec la fréquence maximale du MCU.

Remarque

Si un projet existant est copié dans un autre emplacement, le chemin d'inclusion doit être mis à jour. □

5.2.2 Ajouter le middleware nécessaire au projet (facultatif)

Les piles (ou bibliothèques) middleware disponibles sont:

1. les bibliothèques d'hôtes et périphériques USB,

2. *STemWin*,

3. *LibJPEG*,

4. *FreeRTOS*,

5. *FatFS*,

6. *LwIP*,

7. *PolarSSL.*

Pour savoir quels fichiers source doivent être ajoutés à la liste des fichiers de projet, il faut se référer à la documentation fournie pour chaque middleware. L'utilisateur peut également consulter les applications disponibles sous `/Projects/STM32F4-Discovery/Applications/<MW_Stack>` (`<MW_Stack>` fait référence à la pile middleware, par exemple `USB_Device`) pour avoir une meilleure idée des fichiers sources à ajouter et les chemins d'inclusion.

5.2.3 Configurer les composants du firmware

Les composants HAL et middleware offrent un ensemble d'options de configuration du processus de construction (compilation) à l'aide de macros déclarées avec `#define` dans un fichier d'en-tête. Un fichier de configuration modèle est fourni pour chaque composant; l'utilisateur doit le copier dans le répertoire de son de projet (le fichier de configuration s'appelle `xxx_conf_template.h`) Le mot «`_template`» doit être supprimé lors de sa copie dans le répertoire du projet). Le fichier de configuration fournit suffisamment d'informations pour connaître l'effet de chaque option de configuration. Des informations plus détaillées sont disponibles dans la documentation fournie pour chaque composant.

5.2.4 Démarrer la bibliothèque HAL

Après avoir sauté vers le programme principal, le code d'application doit appeler l'API `HAL_Init()` pour initialiser la bibliothèque HAL, qui effectue les opérations suivantes:

1. Configure les caches d'instruction et de données de pré-lecture de la mémoire flash (configurables par des macros définies dans `stm32f4xx_hal_conf.h`).

2. Configure SysTick pour générer une interruption tous les 1 ms. Le SysTick est cadencé par le HSI (configuration par défaut après la réinitialisation).

3. Définit la priorité du groupe NVIC à 4.

4. Appelle la fonction de rappel (callback) `HAL_MspInit()` définie dans le fichier utilisateur `stm32f4xx_hal_msp.c` pour effectuer l'initialisation matérielle globale de bas niveau.

5.2.5 Configurer l'horloge système

La configuration de l'horloge système se fait en appelant ces deux API:

1. `HAL_RCC_OscConfig()`: configure les oscillateurs internes et/ou externes, la source PLL et les facteurs. L'utilisateur peut choisir de configurer un oscillateur ou tous les oscillateurs. Si le système ne doit pas fonctionner à haute fréquence, l'utilisateur peut sauter la configuration PLL.

2. `HAL_RCC_ClockConfig()`: configure la source de l'horloge système, la latence de la mémoire flash et les pré-sélectionneurs AHB et APB.

5.2.6 Initialisation des périphériques

1. Commencer par écrire la fonction de périphérique `HAL_PPP_MspInit`. Pour cette fonction, procédez comme suit:

    1. Activer l'horloge du périphérique.

    2. Configurer les GPIO du périphériques.

    3. Configurer le canal DMA et activer l'interruption DMA (si nécessaire).

    4. Activer l'interruption du périphérique (si nécessaire).

2. Modifier le fichier `stm32f4xx_it.c` pour appeler les gestionnaires d'interruptions nécessaires (périphériques et DMA), si nécessaire.

3. Écrire les fonctions de rappel (callback functions) complètes si on envisage d'utiliser l'interruption périphérique ou le DMA.

4. Dans le fichier main.c, initialiser la structure du handle de périphérique, puis appeler la fonction `HAL_PPP_Init()` pour initialiser le périphérique.

5.2.7 Développer le code de l'application

Développer un processus d'application: à ce stade, le système est prêt et l'utilisateur peut commencer à développer le code de l'application.

1. Le HAL fournit des APIs intuitives et prêtes à l'emploi pour configurer le périphérique et prend en charge les modèles de programmation par interrogation, par interruption et de par DMA, pour répondre à toutes les exigences de l'application. Pour plus de détails sur l'utilisation de chaque périphérique, se référer au riche ensemble d'exemples fourni.

2. Si l'application a certaines contraintes en temps réel, l'utilisateur peut trouver un grand ensemble d'exemples montrant comment utiliser le FreeRTOS et l'intégrer avec les piles middleware fournies dans le STM32CubeF4. Ce peut être un bon point de départ pour un premier développement.

Remarque

Dans l'implémentation par défaut de HAL, le timer SysTick est la source de la base de temps. Il est utilisé pour générer des interruptions à des intervalles de temps réguliers. Si `HAL_Delay ()` est appelée à partir d'un processus ISR de périphérique, l'interruption SysTick doit avoir une priorité plus élevée (numériquement inférieure) que l'interruption du périphérique. Sinon, le processus ISR de l'appelant est bloqué. Les fonctions qui affectent les configurations de la base de temps sont déclarées comme `__Weak` pour rendre la substitution possible dans le cas d'autres implémentations dans le fichier utilisateur (à l'aide, par exemple, d'un timer à usage général ou d'une autre source de temps). Pour plus de détails, se référer à l'exemple `HAL_TimeBase`. □

5.3 Utilisation de STM32CubeMX pour générer le code C d'initialisation

Une autre alternative aux étapes 1 à 6 décrites dans la section 5.2 consiste à utiliser l'outil *STM32CubeMX* pour générer facilement un code pour l'initialisation du système, des périphériques et du middleware:

1. Sélection du microcontrôleur STMicroelectronics STM32 qui correspond à l'ensemble des périphériques requis.

2. Configuration de chaque logiciel embarqué requis grâce à un solveur de conflits de broches, un assistant de paramétrage d'horloge, un calculateur de consommation d'énergie et un utilitaire exécutant une configuration des périphériques du MCU (GPIO, USART ...) et des piles middleware (USB, TCP/IP ...).

3. Génération du code C d'initialisation en fonction de la configuration sélectionnée. Ce code est prêt à être utilisé dans plusieurs environnements de développement. Le code utilisateur est laissé à la génération de code suivante.

   Pour plus d'informations, reportez-vous au manuel d'utilisation de STMCubeMX pour STM32 [2].

5.4 Comment obtenir les mises à jour de STM32CubeF4

Le progiciel STM32CubeF4 est livré avec un utilitaire de mise à jour: *STM32CubeUpdater*, également disponible sous forme de menu dans l'outil de génération de code STM32CubeMX.

STM32CubeUpdater détecte les nouvelles versions de firmware et les correctifs disponibles sur le site de STMicroelectronics et propose de les télécharger sur l'ordinateur de l'utilisateur.

6 La FAQ

6.1 Existe-t-il un lien avec les bibliothèques de périphériques standard?

La couche HAL de STM32Cube remplace la bibliothèque standard de périphériques.

Les APIs HAL offrent un niveau d'abstraction plus élevé par rapport aux APIs de périphériques standard. HAL se concentre sur les fonctionnalités communes du périphérique plutôt que sur le matériel. Le niveau d'abstraction plus élevé permet de définir un ensemble d'APIs conviviales qui peuvent être facilement portées d'un produit à un autre.

Les bibliothèques de périphériques standard existantes demeurent prises en charge, mais ne sont pas recommandées pour les nouvelles conceptions.

6.2 Le HAL profite-t-il des interruptions ou du DMA? Comment contrôler cela?

Oui. Le HAL prend en charge trois modèles de programmation: interrogation, interruption et DMA (avec ou sans génération d'interruption).

6.3 Comment gère-t-on les caractéristiques spécifiques aux produits/périphériques?

Le HAL offre des API étendues, c'est-à-dire des fonctions spécifiques en tant que modules complémentaires à l'API commune pour prendre en charge

les fonctionnalités disponibles sur certains produits/périphériques uniquement.

6.4 Comment STM32CubeMX peut-il générer du code basé sur un logiciel embarqué?

STM32CubeMX a une connaissance intégrée des microcontrôleurs STM32, y compris leurs périphériques et logiciels. Cela permet à l'outil de fournir une représentation graphique à l'utilisateur et de générer des fichiers `.h` / `.c` en fonction de la configuration choisie par l'utilisateur.

Références

[1] *STMicroelectronics*
www.st.com

[2] "*STM32CubeMX for STM32 configuration and initialization C code generation*" user manual (UM1718)
http://www.st.com/resource/en/user_manual/dm00104712.pdf