IN THE UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF TEXAS
DALLAS DIVISION

LUCIO DEVELOPMENT LLC,	§	
	§	
Plaintiff,	§	Case No:
	§	
vs.	§	PATENT CASE
	§	
STMICROELECTRONICS, INC.,	§	
	§	
Defendant.	§	
_____	§	

COMPLAINT

Plaintiff Lucio Development LLC ("Plaintiff" or "Lucio") files this Complaint against STMicroelectronics, Inc. ("Defendant" or "STM") for infringement of United States Patent No. 7,069,546 (hereinafter "the '546 Patent").

PARTIES AND JURISDICTION

1. This is an action for patent infringement under Title 35 of the United States Code. Plaintiff is seeking injunctive relief as well as damages.

2. Jurisdiction is proper in this Court pursuant to 28 U.S.C. §§ 1331 (Federal Question) and 1338(a) (Patents) because this is a civil action for patent infringement arising under the United States patent statutes.

3. Plaintiff is a Texas limited liability company with its office address at 555 Republic Dr., Suite 200, Plano, Texas 75074.

4. On information and belief, Defendant is a Delaware corporation with its principal place of business located at 750 Canyon Drive, Suite 300, Coppell, TX 75019. On information and belief, Defendant is registered to do business in the State of Texas and can be

served with process upon its registered agent, CT Corp System at 1999 Bryan St., Ste. 900, Dallas, Texas 75201-3136.

5. This Court has personal jurisdiction over Defendant because Defendant has committed, and continues to commit, acts of infringement in this District, has conducted business in this District, and/or has engaged in continuous and systematic activities in this District.

6. On information and belief, Defendant's instrumentalities that are alleged herein to infringe were and continue to be used, imported, offered for sale, and/or sold in this District.

VENUE

7. Venue is proper in this District pursuant to 28 U.S.C. §1400(b) because acts of infringement are occurring in this District and Defendant has a regular and established place of business in this District. For instance, on information and belief, Defendant has a regular and established place of business at 750 Canyon Drive, Suite 300, Coppell, TX 75019.

COUNT I
(INFRINGEMENT OF UNITED STATES PATENT NO. 7,069,546)

8. Plaintiff incorporates paragraphs 1 through 7 herein by reference.

9. This cause of action arises under the patent laws of the United States and, in particular, under 35 U.S.C. §§ 271, *et seq*.

10. Plaintiff is the owner by assignment of the '546 Patent with sole rights to enforce the '546 Patent and sue infringers.

11. A copy of the '546 Patent, titled "Generic Framework for Embedded Software Development," is attached hereto as Exhibit A.

12. The '546 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code.

13. On information and belief, Defendant has infringed and continues to infringe one or more claims, including at least Claim 1, of the '546 Patent by making, using, importing, selling, and/or offering for sale a software platform for embedded software development, which is covered by at least Claim 1 of the '546 Patent. Defendant has infringed and continues to infringe the '546 Patent directly in violation of 35 U.S.C. § 271.

14. Defendant, sells, offers to sell, and/or uses embedded software packages including, without limitation, STM32Cube, and any similar products ("Product"), which infringe at least Claim 1 of the '546 Patent.

15. The Product is a framework that is configured to create embedded software for multiple hardware modules (e.g., versions of a microcontroller, such as the STM32). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

STM32Cube™ is an STMicroelectronics original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube covers STM32 portfolio.

STM32Cube includes the STM32CubeMX which is a graphical software configuration tool that allows generating C initialization code using graphical wizards.

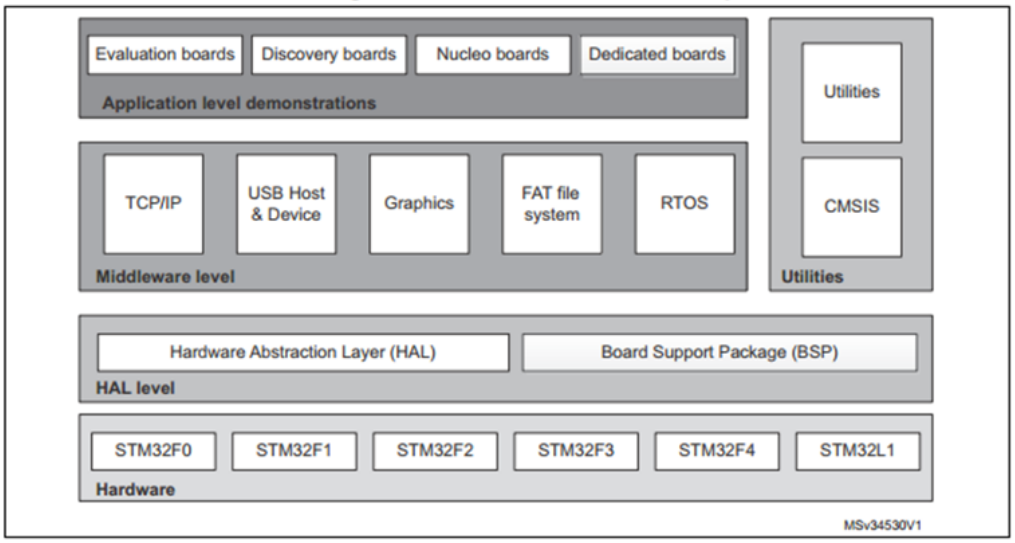
It also comprises the STM32CubeF2 platform which includes the STM32Cube HAL (an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio), plus a consistent set of middleware components (RTOS, USB, TCP/IP and graphics). All embedded software utilities come with a full set of examples.

STM32CubeF2 gathers in one single package all the generic embedded software components required to develop an application on STM32F2 microcontrollers. Following STM32Cube initiative, this set of components is highly portable, not only within STM32F2 series but also to other STM32 series.

STM32CubeF2 is fully compatible with STM32CubeMX code generator that allows generating initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for user convenience.

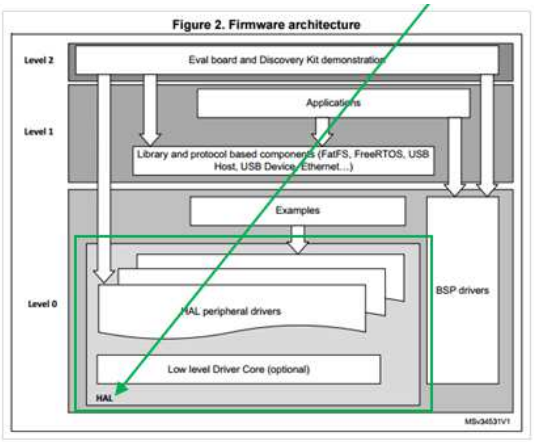
Source: Page 1, http://www.st.com/content/ccc/resource/technical/document/data_brief/b9/7f/2d/41/c4/cc/44/81/DM00110364.pdf/files/DM00110364.pdf/jcr:content/translations/en.DM00110364.pdf

Figure 1. STM32CubeF2 firmware components



Source: Page 5, http://www.st.com/content/ccc/resource/technical/document/user_manual/71/4c/6a/cc/cc/a0/4e/85/DM00111485.pdf/files/DM00111485.pdf/jcr:content/translations/en.DM00111485.pdf

16. The Product includes a Hardware Abstraction Layer (HAL) that provides multiple generic application programming interfaces (APIs). The generic APIs comprise software that provides common and generic functions to multiple hardware modules (e.g., versions of a microcontroller, such as the STM32). These modules may be used in a communication system. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



Hardware Abstraction Layer (HAL): this layer provides the low-level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides a generic, multi instance and function-oriented APIs which allow to offload the user application implementation by providing ready-to-use processes. As an example, for the communication peripherals (I2S, UART...) it

provides APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and manage communication errors that may raise during communication. The HAL Drivers APIs are split into two categories:

- the generic APIs that provide common and generic functions to all the STM32 series,
- the extension APIs that provide specific and customized functions for a specific family or a specific part number.

STM32Cube includes the STM32CubeMX which is a graphical software configuration tool that allows generating C initialization code using graphical wizards.

Source: Pages 6 and 7, http://www.st.com/content/ccc/resource/technical/document/user_manual/71/4c/6a/cc/cc/a0/4e/85/DM00111485.pdf/files/DM00111485.pdf/jcr:content/translations/en.DM00111485.pdf; Page 1, http://www.st.com/content/ccc/resource/technical/document/data_brief/b9/7f/2d/41/c4/cc/44/81/DM00110364.pdf/files/DM00110364.pdf/jcr:content/translations/en.DM00110364.pdf

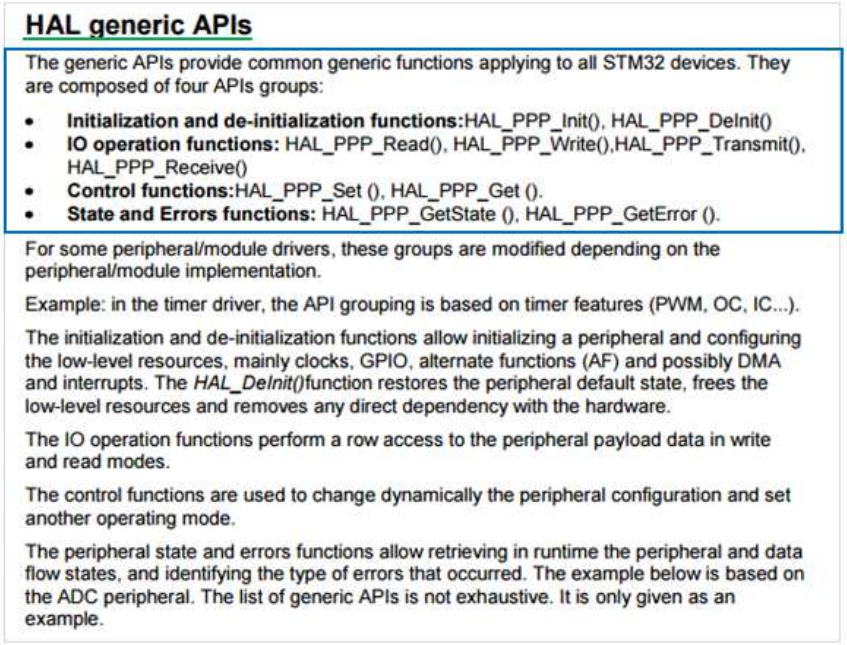
The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees easy portability to other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

The HAL driver APIs are split into two categories: generic APIs which provide common and generic functions for all the STM32 series and extension APIs which include specific and customized functions for a given family or part number.

The HAL drivers are feature-oriented instead of IP-oriented. As an example, the timer APIs are split into several categories following the functions offered by the IP: basic timer, capture, pulse width modulation (PWM), etc..

Source: Page 1, http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf



HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions:** HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions:**HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions:** HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (PWM, OC, IC...).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()*function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

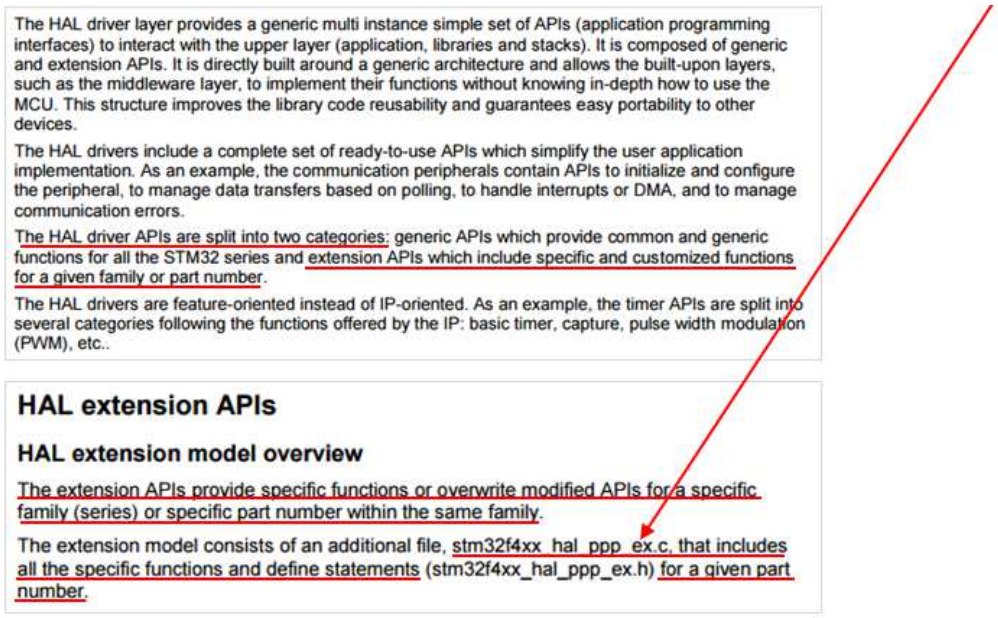
The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in runtime the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Source: Page 40, http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

17. The HAL also provides extension APIs that associate the generic application functions with specific functions for a given hardware module (particular version of a microcontroller). For example, these specific functions are indicated in a file within the extension model: stm32f4xx_hal_ppp_ex.c. This file includes specific functions and define statements. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees easy portability to other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

The HAL driver APIs are split into two categories: generic APIs which provide common and generic functions for all the STM32 series and extension APIs which include specific and customized functions for a given family or part number.

The HAL drivers are feature-oriented instead of IP-oriented. As an example, the timer APIs are split into several categories following the functions offered by the IP: basic timer, capture, pulse width modulation (PWM), etc..

HAL extension APIs

HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32f4xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32f4xx_hal_ppp_ex.h) for a given part number.

Source: Pages 1 and 41, http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

18. The specific APIs include code that defines a specific element in the specific API to be handled by a generic APIC function for at least one of the modules. For instance, the Product's HAL drivers include a specific API that includes a specific element file (e.g., stm32f4xx_hal_ppp_ex.c). This is handled by a generic application function for the hardware module. For example, the generic API includes common functions specified in files (e.g., a main peripheral/module driver file - stm32f4xx_hal_ppp.c, and a header file for the main driver - stm32f4xx_hal_ppp.h. When a specific feature/function needs to be added to a hardware device, the file containing specific functions stm32f4xx_hal_ppp_ex.c is executed along with the file containing generic functions stm32f4xx_hal_ppp.c. Further, the HAL contains data structures that register and embed the required functions. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

HAL drivers are composed of the following set of files:

Table 2: HAL driver files

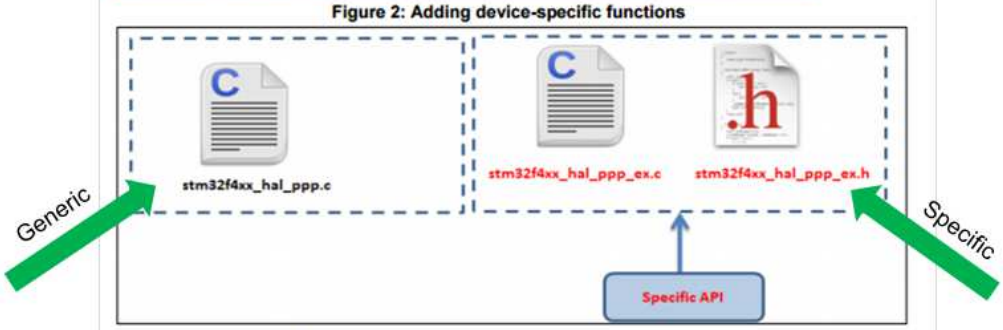
File	Description
stm32f4xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f4xx_hal_adc.c, stm32f4xx_hal_irda.c, ...*
stm32f4xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example: stm32f4xx_hal_adc.h, stm32f4xx_hal_irda.h, ...*
stm32f4xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example: stm32f4xx_hal_adc_ex.c, stm32f4xx_hal_dma_ex.c, ...*

Source: Pages 28 and 29,
http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

Case1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32f4xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

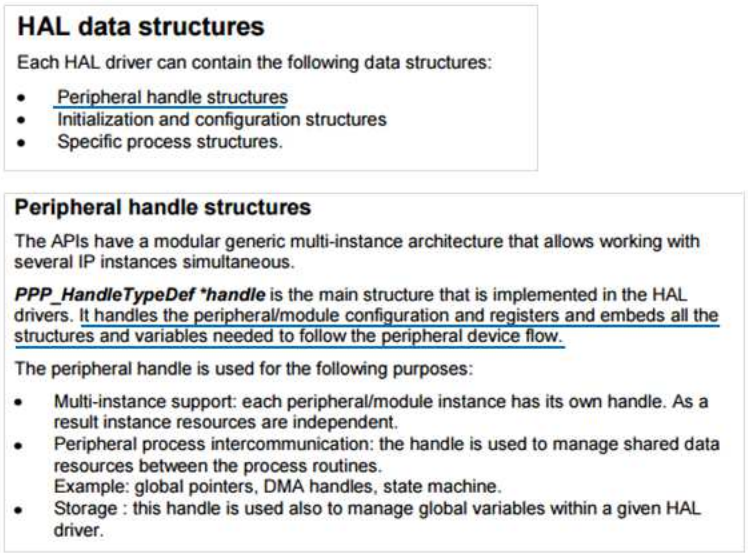
Figure 2: Adding device-specific functions



Example: stm32f4xx_hal_flash_ex.c/h

```
#if defined(STM32F427xx) || defined(STM32F437xx) || defined(STM32F429xx)||
defined(STM32F439xx)
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);
#endif /* STM32F427xx ||STM32F437xx || STM32F429xx || STM32F439xx  */
```

Source: Page 42,
http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf



HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

Peripheral handle structures

The APIs have a modular generic multi-instance architecture that allows working with several IP instances simultaneous.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi-instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
  Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

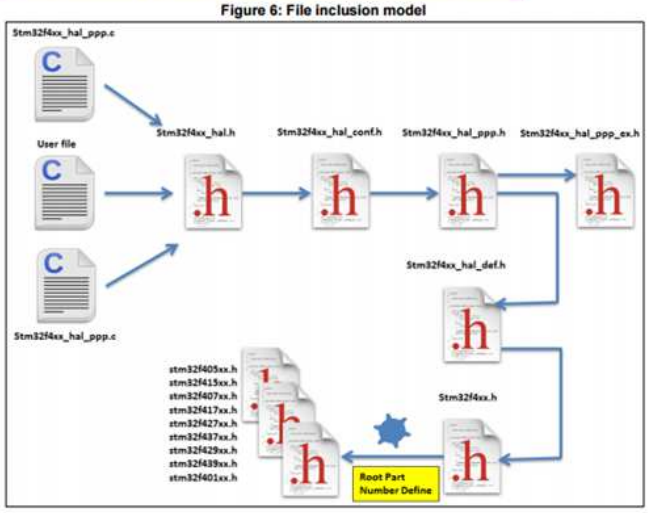
Source: Pages 31 and 32,
http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

19. When a specific API is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. This can be executed by an embedded processor in the hardware module. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

File inclusion model

The header of the common HAL driver file (stm32f4xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6: File inclusion model

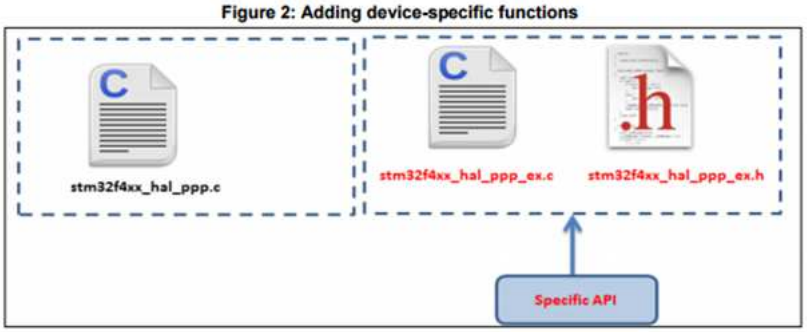


Source: Page 44, http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

Case1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32f4xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2: Adding device-specific functions



Example: stm32f4xx_hal_flash_ex.c/h

```
#if defined(STM32F427xx) || defined(STM32F437xx) || defined(STM32F429xx)||
defined(STM32F439xx)
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);
#endif /* STM32F427xx ||STM32F437xx || STM32F429xx || STM32F439xx  */
```

Source: Page 42, http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

20. Defendant's actions complained of herein will continue unless Defendant is enjoined by this court.

21. Defendant's actions complained of herein are causing irreparable harm and monetary damage to Plaintiff and will continue to do so unless and until Defendant is enjoined

and restrained by this Court.

22. Plaintiff is in compliance with 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff asks the Court to:

(a) Enter judgment for Plaintiff on this Complaint on all causes of action asserted herein;

(b) Enter an Order enjoining Defendant, its agents, officers, servants, employees, attorneys, and all persons in active concert or participation with Defendant who receive notice of the order from further infringement of United States Patent No. 7,069,546 (or, in the alternative, awarding Plaintiff a running royalty from the time of judgment going forward);

(c) Award Plaintiff damages resulting from Defendant's infringement in accordance with 35 U.S.C. § 284;

(d) Award Plaintiff pre-judgment and post-judgment interest and costs; and

(e) Award Plaintiff such further relief to which the Court finds Plaintiff entitled under law or equity.

Dated: December 12, 2017

Respectfully submitted,

/s/ Jay Johnson

JAY JOHNSON
State Bar No. 24067322
D. BRADLEY KIZZIA
State Bar No. 11547550
KIZZIA JOHNSON, PLLC
1910 Pacific Ave., Suite 13000
Dallas, Texas 75201
(214) 451-0164
Fax: (214) 451-0165
jay@kjpllc.com
bkizzia@kjpllc.com

ATTORNEYS FOR PLAINTIFF

EXHIBIT A