Low Voltage Variable Frequency Drive

2.1

Generado por Doxygen 1.15.0

Capítulo 1

Lista de tareas pendientes

Global GestorEstados_Action (SystemAction sysAct, int value)

: Aca tengo que poner un timer para que calcule el tiempo aprox de frenado.

: Revisar si es necesario chequear esto ¿Por qué no ejecutamos el stop sin importar el estado?

Capítulo 2

Topic Index

2.1. Topics

Here is a list of all topics with brief descriptions:

Pines del inversor (puerto A) . . . . . **??**
Pines de señalización (puerto B) . . . . . **??**
CMSIS . . . . . **??**
Stm32f1xx_system . . . . . **??**
STM32F1xx_System_Private_Includes . . . . . **??**
STM32F1xx_System_Private_TypesDefinitions . . . . . **??**
STM32F1xx_System_Private_Defines . . . . . **??**
STM32F1xx_System_Private_Macros . . . . . **??**
STM32F1xx_System_Private_Variables . . . . . **??**
STM32F1xx_System_Private_FunctionPrototypes . . . . . **??**
STM32F1xx_System_Private_Functions . . . . . **??**

Capítulo 3

Jerarquía de directorios

3.1. Directorios

Gestor_Estados . . . . . ??
GestorEstados.c . . . . . ??
GestorEstados.h . . . . . ??
Gestor_SVM . . . . . ??
GestorSVM.c . . . . . ??
GestorSVM.h . . . . . ??
Gestor_Timers . . . . . ??
GestorTimers.c . . . . . ??
GestorTimers.h . . . . . ??
Inc . . . . . ??
main.h . . . . . ??
stm32f1xx_hal_conf.h . . . . . ??
stm32f1xx_it.h . . . . . ??
Modules . . . . . ??
Gestor_Estados . . . . . ??
GestorEstados.c . . . . . ??
GestorEstados.h . . . . . ??
Gestor_SVM . . . . . ??
GestorSVM.c . . . . . ??
GestorSVM.h . . . . . ??
Gestor_Timers . . . . . ??
GestorTimers.c . . . . . ??
GestorTimers.h . . . . . ??
SPI_Interfase . . . . . ??
SPIModule.c . . . . . ??
SPIModule.h . . . . . ??
SPI_Interfase . . . . . ??
SPIModule.c . . . . . ??
SPIModule.h . . . . . ??
Src . . . . . ??
main.c . . . . . ??
stm32f1xx_hal_msp.c . . . . . ??
stm32f1xx_it.c . . . . . ??
syscalls.c . . . . . ??
sysmem.c . . . . . ??
system_stm32f1xx.c . . . . . ??

Capítulo 4

Índice de estructuras de datos

4.1. Estructuras de datos

Lista de estructuras con breves descripciones:

ConfiguracionSVM
: Parámetros de configuración del módulo SVM . . . . . **??**

DatoCalculado . . . . . **??**

Parametros
: Parámetros “sombra” para sincronización atómica con el lazo de cálculo . . . . . **??**

ValoresSwitching
: Estructura de trabajo del ISR del timer de switching . . . . . **??**

Capítulo 5

Índice de archivos

5.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Inc/main.h
Header de la aplicación principal . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Inc/stm32f1xx_hal_conf.h
HAL configuration file . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Inc/stm32f1xx_it.h
This file contains the headers of the interrupt handlers . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/Gestor_Estados/GestorEstados.c
Implementa la máquina de estados del LVFV . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/Gestor_Estados/GestorEstados.h . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/Gestor_SVM/GestorSVM.c . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/Gestor_SVM/GestorSVM.h . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/Gestor_Timers/GestorTimers.c . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/Gestor_Timers/GestorTimers.h . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/SPI_Interfase/SPIModule.c . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Modules/SPI_Interfase/SPIModule.h
Interfaz del módulo SPI (esclavo) con DMA . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Src/main.c
Punto de entrada del firmware y configuración de periféricos . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Src/stm32f1xx_hal_msp.c
This file provides code for the MSP Initialization and de-Initialization codes . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩ Src/stm32f1xx_it.c
Interrupt Service Routines . . . . . ??

C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩
Src/syscalls.c
STM32CubeIDE Minimal System calls file . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩
Src/sysmem.c
STM32CubeIDE System Memory calls file . . . . . ??
C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/↩
Src/system_stm32f1xx.c
CMSIS Cortex-M3 Device Peripheral Access Layer System Source File . . . . . ??

Capítulo 6

Topic Documentation

6.1. Pines del inversor (puerto A)

Pines de control del módulo SVM en GPIOA.

defines

- #define GPIO_U_IN GPIO_PIN_1
  Entrada lógica de la pierna U (GPIOA, PIN_1).
- #define GPIO_U_SD GPIO_PIN_2
  Shutdown/Habilitación driver pierna U (GPIOA, PIN_2).
- #define GPIO_V_IN GPIO_PIN_3
  Entrada lógica de la pierna V (GPIOA, PIN_3).
- #define GPIO_V_SD GPIO_PIN_4
  Shutdown/Habilitación driver pierna V (GPIOA, PIN_4).
- #define GPIO_W_IN GPIO_PIN_5
  Entrada lógica de la pierna W (GPIOA, PIN_5).
- #define GPIO_W_SD GPIO_PIN_6
  Shutdown/Habilitación driver pierna W (GPIOA, PIN_6).
- #define GPIO_TERMO_SWITCH GPIO_PIN_11
  Entrada digital: Termo–switch (GPIOA, PIN_11).
- #define GPIO_STOP_BUTTON GPIO_PIN_12
  Entrada digital: Botón de parada (GPIOA, PIN_12).

6.1.1. Descripción detallada

Pines de control del módulo SVM en GPIOA.

6.1.2. Documentación de «define»

6.1.2.1. GPIO_STOP_BUTTON

```
#define GPIO_STOP_BUTTON GPIO_PIN_12
```

Entrada digital: Botón de parada (GPIOA, PIN_12).

Definición en la línea 34 del archivo main.h.

6.1.2.2. GPIO_TERMO_SWITCH

```
#define GPIO_TERMO_SWITCH GPIO_PIN_11
```

Entrada digital: Termo–switch (GPIOA, PIN_11).

Definición en la línea 32 del archivo main.h.

6.1.2.3. GPIO_U_IN

```
#define GPIO_U_IN GPIO_PIN_1
```

Entrada lógica de la pierna U (GPIOA, PIN_1).

Definición en la línea 19 del archivo main.h.

6.1.2.4. GPIO_U_SD

```
#define GPIO_U_SD GPIO_PIN_2
```

Shutdown/Habilitación driver pierna U (GPIOA, PIN_2).

Definición en la línea 21 del archivo main.h.

6.1.2.5. GPIO_V_IN

```
#define GPIO_V_IN GPIO_PIN_3
```

Entrada lógica de la pierna V (GPIOA, PIN_3).

Definición en la línea 23 del archivo main.h.

6.1.2.6. GPIO_V_SD

```
#define GPIO_V_SD GPIO_PIN_4
```

Shutdown/Habilitación driver pierna V (GPIOA, PIN_4).

Definición en la línea 25 del archivo main.h.

6.1.2.7. GPIO_W_IN

```
#define GPIO_W_IN GPIO_PIN_5
```

Entrada lógica de la pierna W (GPIOA, PIN_5).

Definición en la línea 27 del archivo main.h.

6.1.2.8. GPIO_W_SD

```
#define GPIO_W_SD GPIO_PIN_6
```

Shutdown/Habilitación driver pierna W (GPIOA, PIN_6).

Definición en la línea 29 del archivo main.h.

6.2. Pines de señalización (puerto B)

LEDs de estado en GPIOB.

defines

- #define GPIO_LED_STATE GPIO_PIN_0
  LED de estado (GPIOB, PIN_0).
- #define GPIO_LED_ERROR GPIO_PIN_2
  LED de error (GPIOB, PIN_2).

6.2.1. Descripción detallada

LEDs de estado en GPIOB.

6.2.2. Documentación de «define»

6.2.2.1. GPIO_LED_ERROR

```
#define GPIO_LED_ERROR GPIO_PIN_2
```

LED de error (GPIOB, PIN_2).

Definición en la línea 45 del archivo main.h.

6.2.2.2. GPIO_LED_STATE

```
#define GPIO_LED_STATE GPIO_PIN_0
```

LED de estado (GPIOB, PIN_0).

Definición en la línea 43 del archivo main.h.

6.3. CMSIS

Diagrama de colaboración de CMSIS:

Topics

- Stm32f1xx_system . . . . . ??

6.3.1. Descripción detallada

6.3.2. Stm32f1xx_system

Diagrama de colaboración de Stm32f1xx_system:

Topics

- STM32F1xx_System_Private_Includes . . . . . ??
- STM32F1xx_System_Private_TypesDefinitions . . . . . ??
- STM32F1xx_System_Private_Defines . . . . . ??
- STM32F1xx_System_Private_Macros . . . . . ??
- STM32F1xx_System_Private_Variables . . . . . ??
- STM32F1xx_System_Private_FunctionPrototypes . . . . . ??
- STM32F1xx_System_Private_Functions . . . . . ??

6.3.2.1. Descripción detallada

6.3.2.2. STM32F1xx_System_Private_Includes

Diagrama de colaboración de STM32F1xx_System_Private_Includes:

6.3.2.3. STM32F1xx_System_Private_TypesDefinitions

Diagrama de colaboración de STM32F1xx_System_Private_TypesDefinitions:

6.3.2.4. STM32F1xx_System_Private_Defines

Diagrama de colaboración de STM32F1xx_System_Private_Defines:

defines

- #define HSE_VALUE 8000000U
- #define HSI_VALUE 8000000U

6.3.2.4.1. Descripción detallada

6.3.2.4.2. Documentación de «define»

6.3.2.4.2.1. HSE_VALUE

```
#define HSE_VALUE 8000000U
```

Default value of the External oscillator in Hz. This value can be provided and adapted by the user application.

Definición en la línea 77 del archivo system_stm32f1xx.c.

6.3.2.4.2.2. HSI_VALUE

```
#define HSI_VALUE 8000000U
```

Default value of the Internal oscillator in Hz. This value can be provided and adapted by the user application.

Definición en la línea 82 del archivo system_stm32f1xx.c.

6.3.2.5. STM32F1xx_System_Private_Macros

Diagrama de colaboración de STM32F1xx_System_Private_Macros:

6.3.2.6. STM32F1xx_System_Private_Variables

Diagrama de colaboración de STM32F1xx_System_Private_Variables:

Variables

- uint32_t SystemCoreClock = 8000000
- const uint8_t AHBPrescTable [16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t APBPrescTable [8U] = {0, 0, 0, 0, 1, 2, 3, 4}

6.3.2.6.1. Descripción detallada

6.3.2.6.2. Documentación de variables

6.3.2.6.2.1. AHBPrescTable

```
const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
```

Definición en la línea 142 del archivo system_stm32f1xx.c.

6.3.2.6.2.2. APBPrescTable

```
const uint8_t APBPrescTable[8U] = {0, 0, 0, 0, 1, 2, 3, 4}
```

Definición en la línea 143 del archivo system_stm32f1xx.c.

6.3.2.6.2.3. SystemCoreClock

```
uint32_t SystemCoreClock = 8000000
```

Definición en la línea 141 del archivo system_stm32f1xx.c.

6.3.2.7. STM32F1xx_System_Private_FunctionPrototypes

Diagrama de colaboración de STM32F1xx_System_Private_FunctionPrototypes:

6.3.2.8. STM32F1xx_System_Private_Functions

Diagrama de colaboración de STM32F1xx_System_Private_Functions:

Funciones

- void SystemInit (void)

  Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

6.3.2.8.1. Descripción detallada

6.3.2.8.2. Documentación de funciones

6.3.2.8.2.1. SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
            void )
```

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Nota

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(∗)
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(∗∗)
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(∗∗) or HSI_VALUE(∗) multiplied by the PLL factors.

(∗) HSI_VALUE is a constant defined in stm32f1xx.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature.

(∗∗) HSE_VALUE is a constant defined in stm32f1xx.h file (default value 8 MHz or 25 MHz, depending on the product used), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

- The result of this function could be not correct when using fractional value for HSE crystal.

Parámetros

None	

Valores devueltos

None	

Definición en la línea 224 del archivo system_stm32f1xx.c.

6.3.2.8.2.2. SystemInit()

```
void SystemInit (
             void )
```

Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.

Nota

This function should be used only after reset.

Parámetros

None	

Valores devueltos

None	

Definición en la línea 175 del archivo system_stm32f1xx.c.

Capítulo 7

Documentación de directorios

7.1. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_Estados

Gráfico de dependencias de directorios de Gestor_Estados:

Archivos

- archivo GestorEstados.c

  Implementa la máquina de estados del LVFV.
- archivo GestorEstados.h

7.2. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_SVM

Gráfico de dependencias de directorios de Gestor_SVM:

Archivos

- archivo GestorSVM.c
- archivo GestorSVM.h

7.3. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_Timers

Gráfico de dependencias de directorios de Gestor_Timers:

Archivos

- archivo GestorTimers.c
- archivo GestorTimers.h

7.4. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Inc

Archivos

- archivo main.h

  Header de la aplicación principal.
- archivo stm32f1xx_hal_conf.h

  HAL configuration file.
- archivo stm32f1xx_it.h

  This file contains the headers of the interrupt handlers.

7.5. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules

Gráfico de dependencias de directorios de Modules:

Directorios

- directorio Gestor_Estados
- directorio Gestor_SVM
- directorio Gestor_Timers
- directorio SPI_Interfase

7.6. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/SPI_Interfase

Gráfico de dependencias de directorios de SPI_Interfase:

Archivos

- archivo SPIModule.c
- archivo SPIModule.h

  Interfaz del módulo SPI (esclavo) con DMA.

7.7. Referencia del directorio C:/Users/User/Desktop/ProyectoFinal-↩ Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src

Gráfico de dependencias de directorios de Src:

Archivos

- archivo main.c

  Punto de entrada del firmware y configuración de periféricos.
- archivo stm32f1xx_hal_msp.c

  This file provides code for the MSP Initialization and de-Initialization codes.
- archivo stm32f1xx_it.c

  Interrupt Service Routines.
- archivo syscalls.c

  STM32CubeIDE Minimal System calls file.
- archivo sysmem.c

  STM32CubeIDE System Memory calls file.
- archivo system_stm32f1xx.c

  CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.

Capítulo 8

Documentación de estructuras de datos

8.1. Referencia de la estructura ConfiguracionSVM

Parámetros de configuración del módulo SVM.

`#include <GestorSVM.h>`

Campos de datos

- int frec_switch
- int frecReferencia

  Frecuencia de switching del PWM SVM [Hz].
- int direccionRotacion

  Frecuencia de referencia (target) en régimen [Hz].
- int acel

  1 = sentido horario, -1 = antihorario.
- int desacel

  Aceleración dinámica [Hz/seg].

8.1.1. Descripción detallada

Parámetros de configuración del módulo SVM.

- frec_switch: frecuencia del timer de switching (Hz).
- frecReferencia: frecuencia objetivo de marcha estable (Hz).
- direccionRotacion: 1 horario, -1 antihorario.
- acel / desacel: rampas dinámicas [Hz/seg].

Definición en la línea 31 del archivo GestorSVM.h.

8.1.2. Documentación de campos

8.1.2.1. acel

```
int acel
```

1 = sentido horario, -1 = antihorario.

Definición en la línea 35 del archivo GestorSVM.h.

8.1.2.2. desacel

```
int desacel
```

Aceleración dinámica [Hz/seg].

Definición en la línea 36 del archivo GestorSVM.h.

8.1.2.3. direccionRotacion

```
int direccionRotacion
```

Frecuencia de referencia (target) en régimen [Hz].

Definición en la línea 34 del archivo GestorSVM.h.

8.1.2.4. frec_switch

```
int frec_switch
```

Definición en la línea 32 del archivo GestorSVM.h.

8.1.2.5. frecReferencia

```
int frecReferencia
```

Frecuencia de switching del PWM SVM [Hz].

Definición en la línea 33 del archivo GestorSVM.h.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/↩ Gestor_SVM/GestorSVM.h

8.2. Referencia de la estructura DatoCalculado

Campos de datos

- int ticksChannel1 [BUFFER_CALCULO_SIZE]
- int ticksChannel2 [BUFFER_CALCULO_SIZE]
- int ticksChannel3 [BUFFER_CALCULO_SIZE]
- CasoInterferenciaTimer casoInterferencia [BUFFER_CALCULO_SIZE]
- int cuadranteActual [BUFFER_CALCULO_SIZE]
- int indiceEscritura
- int indiceLectura

  Índice de escritura del productor.
- int contadorDeDatos

  Índice de lectura del consumidor.

8.2.1. Descripción detallada

Definición en la línea 134 del archivo GestorSVM.c.

8.2.2. Documentación de campos

8.2.2.1. casoInterferencia

```
CasoInterferenciaTimer casoInterferencia[BUFFER_CALCULO_SIZE]
```

Definición en la línea 138 del archivo GestorSVM.c.

8.2.2.2. contadorDeDatos

```
int contadorDeDatos
```

Índice de lectura del consumidor.

Definición en la línea 142 del archivo GestorSVM.c.

8.2.2.3. cuadranteActual

```
int cuadranteActual[BUFFER_CALCULO_SIZE]
```

Definición en la línea 139 del archivo GestorSVM.c.

8.2.2.4. indiceEscritura

```
int indiceEscritura
```

Definición en la línea 140 del archivo GestorSVM.c.

8.2.2.5. indiceLectura

```
int indiceLectura
```

Índice de escritura del productor.

Definición en la línea 141 del archivo GestorSVM.c.

8.2.2.6. ticksChannel1

```
int ticksChannel1[BUFFER_CALCULO_SIZE]
```

Definición en la línea 135 del archivo GestorSVM.c.

8.2.2.7. ticksChannel2

```
int ticksChannel2[BUFFER_CALCULO_SIZE]
```

Definición en la línea 136 del archivo GestorSVM.c.

8.2.2.8. ticksChannel3

```
int ticksChannel3[BUFFER_CALCULO_SIZE]
```

Definición en la línea 137 del archivo GestorSVM.c.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/↩ Gestor_SVM/GestorSVM.c

8.3. Referencia de la estructura Parametros

Parámetros “sombra” para sincronización atómica con el lazo de cálculo.

Campos de datos

- int32_t frecObjetivo
- uint32_t cambioFrecuenciaPorCiclo
- int direccionRotacion
- int flagMotorRunning
- int flagEsAcelerado
- int flagChangingFrecuencia

8.3.1. Descripción detallada

Parámetros “sombra” para sincronización atómica con el lazo de cálculo.

Definición en la línea 150 del archivo GestorSVM.c.

8.3.2. Documentación de campos

8.3.2.1. cambioFrecuenciaPorCiclo

```
uint32_t cambioFrecuenciaPorCiclo
```

Definición en la línea 152 del archivo GestorSVM.c.

8.3.2.2. direccionRotacion

```
int direccionRotacion
```

Definición en la línea 153 del archivo GestorSVM.c.

8.3.2.3. flagChangingFrecuencia

```
int flagChangingFrecuencia
```

Definición en la línea 156 del archivo GestorSVM.c.

8.3.2.4. flagEsAcelerado

```
int flagEsAcelerado
```

Definición en la línea 155 del archivo GestorSVM.c.

8.3.2.5. flagMotorRunning

```
int flagMotorRunning
```

Definición en la línea 154 del archivo GestorSVM.c.

8.3.2.6. frecObjetivo

```
int32_t frecObjetivo
```

Definición en la línea 151 del archivo GestorSVM.c.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/↩ Gestor_SVM/GestorSVM.c

8.4. Referencia de la estructura ValoresSwitching

Estructura de trabajo del ISR del timer de switching.

`#include <GestorSVM.h>`

Campos de datos

- char cuadrante
- char flagSwitch [3]

  Sector SVM actual (0..5). Lo usa el ISR para decidir qué pierna conmutar.
- int ticks1 [2]

  Flags de acción por pierna {U,V,W}: 1=encender, 0=apagar (par de valores por flanco).
- int ticks2 [2]

  Ticks CCR del evento 1 (subida/bajada).
- int ticks3 [2]

  Ticks CCR del evento 2 (subida/bajada).
- int contador

  Ticks CCR del evento 3 (subida/bajada).

8.4.1. Descripción detallada

Estructura de trabajo del ISR del timer de switching.

Contiene los parámetros que determinan en qué puntos (ticks) se disparan las comparaciones CCR para conmutar las fases y el estado de conmutación (encendido/apagado) de cada pierna. El campo cuadrante indica el sector SVM actual (0..5).

Definición en la línea 13 del archivo GestorSVM.h.

8.4.2. Documentación de campos

8.4.2.1. contador

`int contador`

Ticks CCR del evento 3 (subida/bajada).

Definición en la línea 19 del archivo GestorSVM.h.

8.4.2.2. cuadrante

`char cuadrante`

Definición en la línea 14 del archivo GestorSVM.h.

8.4.2.3. flagSwitch

```
char flagSwitch[3]
```

Sector SVM actual (0..5). Lo usa el ISR para decidir qué pierna conmutar.

Definición en la línea 15 del archivo GestorSVM.h.

8.4.2.4. ticks1

```
int ticks1[2]
```

Flags de acción por pierna {U,V,W}: 1=encender, 0=apagar (par de valores por flanco).

Definición en la línea 16 del archivo GestorSVM.h.

8.4.2.5. ticks2

```
int ticks2[2]
```

Ticks CCR del evento 1 (subida/bajada).

Definición en la línea 17 del archivo GestorSVM.h.

8.4.2.6. ticks3

```
int ticks3[2]
```

Ticks CCR del evento 2 (subida/bajada).

Definición en la línea 18 del archivo GestorSVM.h.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/↩ Gestor_SVM/GestorSVM.h

Capítulo 9

Documentación de archivos

9.1. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-↩ Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Inc/main.h

Header de la aplicación principal.

```
#include "stm32f1xx_hal.h"
```

Gráfico de dependencias incluidas en main.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

defines

- #define GPIO_U_IN GPIO_PIN_1

  Entrada lógica de la pierna U (GPIOA, PIN_1).
- #define GPIO_U_SD GPIO_PIN_2

  Shutdown/Habilitación driver pierna U (GPIOA, PIN_2).
- #define GPIO_V_IN GPIO_PIN_3

  Entrada lógica de la pierna V (GPIOA, PIN_3).
- #define GPIO_V_SD GPIO_PIN_4

  Shutdown/Habilitación driver pierna V (GPIOA, PIN_4).
- #define GPIO_W_IN GPIO_PIN_5

  Entrada lógica de la pierna W (GPIOA, PIN_5).
- #define GPIO_W_SD GPIO_PIN_6

  Shutdown/Habilitación driver pierna W (GPIOA, PIN_6).
- #define GPIO_TERMO_SWITCH GPIO_PIN_11

  Entrada digital: Termo–switch (GPIOA, PIN_11).
- #define GPIO_STOP_BUTTON GPIO_PIN_12

  Entrada digital: Botón de parada (GPIOA, PIN_12).
- #define GPIO_LED_STATE GPIO_PIN_0

  LED de estado (GPIOB, PIN_0).
- #define GPIO_LED_ERROR GPIO_PIN_2

  LED de error (GPIOB, PIN_2).

Funciones

- void Error_Handler (void)

  Manejador global de errores fatales.

9.1.1. Descripción detallada

Header de la aplicación principal.

Contiene definiciones comunes, asignación de pines y prototipos globales compartidos por todo el proyecto.

Definición en el archivo main.h.

9.1.2. Documentación de funciones

9.1.2.1. Error_Handler()

```
void Error_Handler (
             void )
```

Manejador global de errores fatales.

Manejador genérico de errores.

Detiene la ejecución en un bucle y puede ser utilizado para reportar/diagnosticar condiciones críticas. Implementado en main.c.

Deshabilita IRQs y entra en bucle infinito para provocar reset por IWDG si está activo.

Definición en la línea 375 del archivo main.c.

Gráfico de llamadas a esta función:

9.2. main.h

Ir a la documentación de este archivo.

```
00001
00007
00008 #ifndef __MAIN_H
00009 #define __MAIN_H
00010
00011 #include "stm32f1xx_hal.h"
00012
00017
00019 #define GPIO_U_IN           GPIO_PIN_1
00021 #define GPIO_U_SD           GPIO_PIN_2
00023 #define GPIO_V_IN           GPIO_PIN_3
00025 #define GPIO_V_SD           GPIO_PIN_4
00027 #define GPIO_W_IN           GPIO_PIN_5
00029 #define GPIO_W_SD           GPIO_PIN_6
00030
00032 #define GPIO_TERMO_SWITCH   GPIO_PIN_11
00034 #define GPIO_STOP_BUTTON    GPIO_PIN_12
00036
00041
00043 #define GPIO_LED_STATE      GPIO_PIN_0
00045 #define GPIO_LED_ERROR      GPIO_PIN_2
00047
00054 void Error_Handler(void);
00055
00056 #endif /* __MAIN_H */
```

9.3. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Inc/stm32f1xx_hal_conf.h

HAL configuration file.

```
#include "stm32f1xx_hal_rcc.h"
#include "stm32f1xx_hal_gpio.h"
#include "stm32f1xx_hal_exti.h"
#include "stm32f1xx_hal_dma.h"
#include "stm32f1xx_hal_cortex.h"
#include "stm32f1xx_hal_flash.h"
#include "stm32f1xx_hal_pwr.h"
#include "stm32f1xx_hal_spi.h"
#include "stm32f1xx_hal_tim.h"
#include "stm32f1xx_hal_uart.h"
```

Gráfico de dependencias incluidas en stm32f1xx_hal_conf.h:

defines

- #define HAL_MODULE_ENABLED

  This is the list of modules to be used in the HAL driver.
- #define HAL_DMA_MODULE_ENABLED
- #define HAL_GPIO_MODULE_ENABLED
- #define HAL_SPI_MODULE_ENABLED
- #define HAL_TIM_MODULE_ENABLED
- #define HAL_UART_MODULE_ENABLED
- #define HAL_CORTEX_MODULE_ENABLED
- #define HAL_DMA_MODULE_ENABLED
- #define HAL_FLASH_MODULE_ENABLED
- #define HAL_EXTI_MODULE_ENABLED
- #define HAL_GPIO_MODULE_ENABLED
- #define HAL_PWR_MODULE_ENABLED
- #define HAL_RCC_MODULE_ENABLED
- #define HSE_VALUE 8000000U

  Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).
- #define HSE_STARTUP_TIMEOUT 100U
- #define HSI_VALUE 8000000U

  Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).
- #define LSI_VALUE 40000U

  Internal Low Speed oscillator (LSI) value.
- #define LSE_VALUE 32768U

  External Low Speed oscillator (LSE) value. This value is used by the UART, RTC HAL module to compute the system frequency.
- #define LSE_STARTUP_TIMEOUT 5000U
- #define VDD_VALUE 3300U

  This is the HAL system configuration section.
- #define TICK_INT_PRIORITY 15U
- #define USE_RTOS 0U
- #define PREFETCH_ENABLE 1U
- #define USE_HAL_ADC_REGISTER_CALLBACKS 0U /∗ ADC register callback disabled ∗/

- #define USE_HAL_CAN_REGISTER_CALLBACKS 0U /* CAN register callback disabled */
- #define USE_HAL_CEC_REGISTER_CALLBACKS 0U /* CEC register callback disabled */
- #define USE_HAL_DAC_REGISTER_CALLBACKS 0U /* DAC register callback disabled */
- #define USE_HAL_ETH_REGISTER_CALLBACKS 0U /* ETH register callback disabled */
- #define USE_HAL_HCD_REGISTER_CALLBACKS 0U /* HCD register callback disabled */
- #define USE_HAL_I2C_REGISTER_CALLBACKS 0U /* I2C register callback disabled */
- #define USE_HAL_I2S_REGISTER_CALLBACKS 0U /* I2S register callback disabled */
- #define USE_HAL_MMC_REGISTER_CALLBACKS 0U /* MMC register callback disabled */
- #define USE_HAL_NAND_REGISTER_CALLBACKS 0U /* NAND register callback disabled */
- #define USE_HAL_NOR_REGISTER_CALLBACKS 0U /* NOR register callback disabled */
- #define USE_HAL_PCCARD_REGISTER_CALLBACKS 0U /* PCCARD register callback disabled */
- #define USE_HAL_PCD_REGISTER_CALLBACKS 0U /* PCD register callback disabled */
- #define USE_HAL_RTC_REGISTER_CALLBACKS 0U /* RTC register callback disabled */
- #define USE_HAL_SD_REGISTER_CALLBACKS 0U /* SD register callback disabled */
- #define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U /* SMARTCARD register callback disabled */
- #define USE_HAL_IRDA_REGISTER_CALLBACKS 0U /* IRDA register callback disabled */
- #define USE_HAL_SRAM_REGISTER_CALLBACKS 0U /* SRAM register callback disabled */
- #define USE_HAL_SPI_REGISTER_CALLBACKS 0U /* SPI register callback disabled */
- #define USE_HAL_TIM_REGISTER_CALLBACKS 0U /* TIM register callback disabled */
- #define USE_HAL_UART_REGISTER_CALLBACKS 0U /* UART register callback disabled */
- #define USE_HAL_USART_REGISTER_CALLBACKS 0U /* USART register callback disabled */
- #define USE_HAL_WWDG_REGISTER_CALLBACKS 0U /* WWDG register callback disabled */
- #define MAC_ADDR0 2U

  Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.
- #define MAC_ADDR1 0U
- #define MAC_ADDR2 0U
- #define MAC_ADDR3 0U
- #define MAC_ADDR4 0U
- #define MAC_ADDR5 0U
- #define ETH_RX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for receive */
- #define ETH_TX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for transmit */
- #define ETH_RXBUFNB 8U /* 4 Rx buffers of size ETH_RX_BUF_SIZE */
- #define ETH_TXBUFNB 4U /* 4 Tx buffers of size ETH_TX_BUF_SIZE */
- #define DP83848_PHY_ADDRESS 0x01U
- #define PHY_RESET_DELAY 0x000000FFU
- #define PHY_CONFIG_DELAY 0x00000FFFU
- #define PHY_READ_TO 0x0000FFFFU
- #define PHY_WRITE_TO 0x0000FFFFU
- #define PHY_BCR ((uint16_t)0x00)
- #define PHY_BSR ((uint16_t)0x01)
- #define PHY_RESET ((uint16_t)0x8000)
- #define PHY_LOOPBACK ((uint16_t)0x4000)
- #define PHY_FULLDUPLEX_100M ((uint16_t)0x2100)
- #define PHY_HALFDUPLEX_100M ((uint16_t)0x2000)
- #define PHY_FULLDUPLEX_10M ((uint16_t)0x0100)
- #define PHY_HALFDUPLEX_10M ((uint16_t)0x0000)
- #define PHY_AUTONEGOTIATION ((uint16_t)0x1000)
- #define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200)
- #define PHY_POWERDOWN ((uint16_t)0x0800)
- #define PHY_ISOLATE ((uint16_t)0x0400)
- #define PHY_AUTONEGO_COMPLETE ((uint16_t)0x0020)
- #define PHY_LINKED_STATUS ((uint16_t)0x0004)
- #define PHY_JABBER_DETECTION ((uint16_t)0x0002)

- #define PHY_SR ((uint16_t)0x10U)
- #define PHY_SPEED_STATUS ((uint16_t)0x0002U)
- #define PHY_DUPLEX_STATUS ((uint16_t)0x0004U)
- #define USE_SPI_CRC 0U
- #define assert_param(expr)

  Include module's header file.

9.3.1. Descripción detallada

HAL configuration file.

Atención

Definición en el archivo stm32f1xx_hal_conf.h.

9.3.2. Documentación de «define»

9.3.2.1. assert_param

```
#define assert_param(
              expr)
```

Valor:

```
((void)0U)
```

Include module's header file.

Definición en la línea 383 del archivo stm32f1xx_hal_conf.h.

9.3.2.2. DP83848_PHY_ADDRESS

```
#define DP83848_PHY_ADDRESS 0x01U
```

Definición en la línea 188 del archivo stm32f1xx_hal_conf.h.

9.3.2.3. ETH_RX_BUF_SIZE

```
#define ETH_RX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for receive */
```

Definición en la línea 180 del archivo stm32f1xx_hal_conf.h.

9.3.2.4. ETH_RXBUFNB

```
#define ETH_RXBUFNB 8U /* 4 Rx buffers of size ETH_RX_BUF_SIZE */
```

Definición en la línea 182 del archivo stm32f1xx_hal_conf.h.

9.3.2.5. ETH_TX_BUF_SIZE

```
#define ETH_TX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for transmit */
```

Definición en la línea 181 del archivo stm32f1xx_hal_conf.h.

9.3.2.6. ETH_TXBUFNB

```
#define ETH_TXBUFNB 4U /* 4 Tx buffers of size ETH_TX_BUF_SIZE */
```

Definición en la línea 183 del archivo stm32f1xx_hal_conf.h.

9.3.2.7. HAL_CORTEX_MODULE_ENABLED

```
#define HAL_CORTEX_MODULE_ENABLED
```

Definición en la línea 72 del archivo stm32f1xx_hal_conf.h.

9.3.2.8. HAL_DMA_MODULE_ENABLED [1/2]

```
#define HAL_DMA_MODULE_ENABLED
```

Definición en la línea 45 del archivo stm32f1xx_hal_conf.h.

9.3.2.9. HAL_DMA_MODULE_ENABLED [2/2]

```
#define HAL_DMA_MODULE_ENABLED
```

Definición en la línea 45 del archivo stm32f1xx_hal_conf.h.

9.3.2.10. HAL_EXTI_MODULE_ENABLED

```
#define HAL_EXTI_MODULE_ENABLED
```

Definición en la línea 75 del archivo stm32f1xx_hal_conf.h.

9.3.2.11. HAL_FLASH_MODULE_ENABLED

```
#define HAL_FLASH_MODULE_ENABLED
```

Definición en la línea 74 del archivo stm32f1xx_hal_conf.h.

9.3.2.12. HAL_GPIO_MODULE_ENABLED [1/2]

```
#define HAL_GPIO_MODULE_ENABLED
```

Definición en la línea 48 del archivo stm32f1xx_hal_conf.h.

9.3.2.13. HAL_GPIO_MODULE_ENABLED [2/2]

```
#define HAL_GPIO_MODULE_ENABLED
```

Definición en la línea 48 del archivo stm32f1xx_hal_conf.h.

9.3.2.14. HAL_MODULE_ENABLED

```
#define HAL_MODULE_ENABLED
```

This is the list of modules to be used in the HAL driver.

Definición en la línea 36 del archivo stm32f1xx_hal_conf.h.

9.3.2.15. HAL_PWR_MODULE_ENABLED

```
#define HAL_PWR_MODULE_ENABLED
```

Definición en la línea 77 del archivo stm32f1xx_hal_conf.h.

9.3.2.16. HAL_RCC_MODULE_ENABLED

```
#define HAL_RCC_MODULE_ENABLED
```

Definición en la línea 78 del archivo stm32f1xx_hal_conf.h.

9.3.2.17. HAL_SPI_MODULE_ENABLED

```
#define HAL_SPI_MODULE_ENABLED
```

Definición en la línea 65 del archivo stm32f1xx_hal_conf.h.

9.3.2.18. HAL_TIM_MODULE_ENABLED

```
#define HAL_TIM_MODULE_ENABLED
```

Definición en la línea 67 del archivo stm32f1xx_hal_conf.h.

9.3.2.19. HAL_UART_MODULE_ENABLED

```
#define HAL_UART_MODULE_ENABLED
```

Definición en la línea 68 del archivo stm32f1xx_hal_conf.h.

9.3.2.20. HSE_STARTUP_TIMEOUT

```
#define HSE_STARTUP_TIMEOUT 100U
```

Time out for HSE start up, in ms

Definición en la línea 91 del archivo stm32f1xx_hal_conf.h.

9.3.2.21. HSE_VALUE

```
#define HSE_VALUE 8000000U
```

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

Value of the External oscillator in Hz

Definición en la línea 87 del archivo stm32f1xx_hal_conf.h.

9.3.2.22. HSI_VALUE

```
#define HSI_VALUE 8000000U
```

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

Value of the Internal oscillator in Hz

Definición en la línea 100 del archivo stm32f1xx_hal_conf.h.

9.3.2.23. LSE_STARTUP_TIMEOUT

```
#define LSE_STARTUP_TIMEOUT 5000U
```

Time out for LSE start up, in ms

Definición en la línea 121 del archivo stm32f1xx_hal_conf.h.

9.3.2.24. LSE_VALUE

```
#define LSE_VALUE 32768U
```

External Low Speed oscillator (LSE) value. This value is used by the UART, RTC HAL module to compute the system frequency.

< Value of the Internal Low Speed oscillator in Hz The real value may vary depending on the variations in voltage and temperature. Value of the External oscillator in Hz

Definición en la línea 117 del archivo stm32f1xx_hal_conf.h.

9.3.2.25. LSI_VALUE

```
#define LSI_VALUE 40000U
```

Internal Low Speed oscillator (LSI) value.

LSI Typical Value in Hz

Definición en la línea 107 del archivo stm32f1xx_hal_conf.h.

9.3.2.26. MAC_ADDR0

```
#define MAC_ADDR0 2U
```

Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.

Definición en la línea 172 del archivo stm32f1xx_hal_conf.h.

9.3.2.27. MAC_ADDR1

```
#define MAC_ADDR1 0U
```

Definición en la línea 173 del archivo stm32f1xx_hal_conf.h.

9.3.2.28. MAC_ADDR2

```
#define MAC_ADDR2 0U
```

Definición en la línea 174 del archivo stm32f1xx_hal_conf.h.

9.3.2.29. MAC_ADDR3

```
#define MAC_ADDR3 0U
```

Definición en la línea 175 del archivo stm32f1xx_hal_conf.h.

9.3.2.30. MAC_ADDR4

```
#define MAC_ADDR4 0U
```

Definición en la línea 176 del archivo stm32f1xx_hal_conf.h.

9.3.2.31. MAC_ADDR5

```
#define MAC_ADDR5 0U
```

Definición en la línea 177 del archivo stm32f1xx_hal_conf.h.

9.3.2.32. PHY_AUTONEGO_COMPLETE

```
#define PHY_AUTONEGO_COMPLETE ((uint16_t)0x0020)
```

Auto-Negotiation process completed

Definición en la línea 213 del archivo stm32f1xx_hal_conf.h.

9.3.2.33. PHY_AUTONEGOTIATION

```
#define PHY_AUTONEGOTIATION ((uint16_t)0x1000)
```

Enable auto-negotiation function

Definición en la línea 208 del archivo stm32f1xx_hal_conf.h.

9.3.2.34. PHY_BCR

```
#define PHY_BCR ((uint16_t)0x00)
```

Transceiver Basic Control Register

Definición en la línea 199 del archivo stm32f1xx_hal_conf.h.

9.3.2.35. PHY_BSR

```
#define PHY_BSR ((uint16_t)0x01)
```

Transceiver Basic Status Register

Definición en la línea 200 del archivo stm32f1xx_hal_conf.h.

9.3.2.36. PHY_CONFIG_DELAY

```
#define PHY_CONFIG_DELAY 0x00000FFFU
```

Definición en la línea 192 del archivo stm32f1xx_hal_conf.h.

9.3.2.37. PHY_DUPLEX_STATUS

```
#define PHY_DUPLEX_STATUS ((uint16_t)0x0004U)
```

PHY Duplex mask

Definición en la línea 221 del archivo stm32f1xx_hal_conf.h.

9.3.2.38. PHY_FULLDUPLEX_100M

```
#define PHY_FULLDUPLEX_100M ((uint16_t)0x2100)
```

Set the full-duplex mode at 100 Mb/s

Definición en la línea 204 del archivo stm32f1xx_hal_conf.h.

9.3.2.39. PHY_FULLDUPLEX_10M

```
#define PHY_FULLDUPLEX_10M ((uint16_t)0x0100)
```

Set the full-duplex mode at 10 Mb/s

Definición en la línea 206 del archivo stm32f1xx_hal_conf.h.

9.3.2.40. PHY_HALFDUPLEX_100M

```
#define PHY_HALFDUPLEX_100M ((uint16_t)0x2000)
```

Set the half-duplex mode at 100 Mb/s

Definición en la línea 205 del archivo stm32f1xx_hal_conf.h.

9.3.2.41. PHY_HALFDUPLEX_10M

```
#define PHY_HALFDUPLEX_10M ((uint16_t)0x0000)
```

Set the half-duplex mode at 10 Mb/s

Definición en la línea 207 del archivo stm32f1xx_hal_conf.h.

9.3.2.42. PHY_ISOLATE

```
#define PHY_ISOLATE ((uint16_t)0x0400)
```

Isolate PHY from MII

Definición en la línea 211 del archivo stm32f1xx_hal_conf.h.

9.3.2.43. PHY_JABBER_DETECTION

```
#define PHY_JABBER_DETECTION ((uint16_t)0x0002)
```

Jabber condition detected

Definición en la línea 215 del archivo stm32f1xx_hal_conf.h.

9.3.2.44. PHY_LINKED_STATUS

```
#define PHY_LINKED_STATUS ((uint16_t)0x0004)
```

Valid link established

Definición en la línea 214 del archivo stm32f1xx_hal_conf.h.

9.3.2.45. PHY_LOOPBACK

```
#define PHY_LOOPBACK ((uint16_t)0x4000)
```

Select loop-back mode

Definición en la línea 203 del archivo stm32f1xx_hal_conf.h.

9.3.2.46. PHY_POWERDOWN

```
#define PHY_POWERDOWN ((uint16_t)0x0800)
```

Select the power down mode

Definición en la línea 210 del archivo stm32f1xx_hal_conf.h.

9.3.2.47. PHY_READ_TO

```
#define PHY_READ_TO 0x0000FFFFU
```

Definición en la línea 194 del archivo stm32f1xx_hal_conf.h.

9.3.2.48. PHY_RESET

```
#define PHY_RESET ((uint16_t)0x8000)
```

PHY Reset

Definición en la línea 202 del archivo stm32f1xx_hal_conf.h.

9.3.2.49. PHY_RESET_DELAY

```
#define PHY_RESET_DELAY 0x000000FFU
```

Definición en la línea 190 del archivo stm32f1xx_hal_conf.h.

9.3.2.50. PHY_RESTART_AUTONEGOTIATION

```
#define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200)
```

Restart auto-negotiation function

Definición en la línea 209 del archivo stm32f1xx_hal_conf.h.

9.3.2.51. PHY_SPEED_STATUS

```
#define PHY_SPEED_STATUS ((uint16_t)0x0002U)
```

PHY Speed mask

Definición en la línea 220 del archivo stm32f1xx_hal_conf.h.

9.3.2.52. PHY_SR

```
#define PHY_SR ((uint16_t)0x10U)
```

PHY status register Offset

Definición en la línea 218 del archivo stm32f1xx_hal_conf.h.

9.3.2.53. PHY_WRITE_TO

```
#define PHY_WRITE_TO 0x0000FFFFU
```

Definición en la línea 195 del archivo stm32f1xx_hal_conf.h.

9.3.2.54. PREFETCH_ENABLE

```
#define PREFETCH_ENABLE 1U
```

Definición en la línea 134 del archivo stm32f1xx_hal_conf.h.

9.3.2.55. TICK_INT_PRIORITY

```
#define TICK_INT_PRIORITY 15U
```

tick interrupt priority (lowest by default)

Definición en la línea 132 del archivo stm32f1xx_hal_conf.h.

9.3.2.56. USE_HAL_ADC_REGISTER_CALLBACKS

```
#define USE_HAL_ADC_REGISTER_CALLBACKS 0U /* ADC register callback disabled */
```

Definición en la línea 136 del archivo stm32f1xx_hal_conf.h.

9.3.2.57. USE_HAL_CAN_REGISTER_CALLBACKS

```
#define USE_HAL_CAN_REGISTER_CALLBACKS 0U /* CAN register callback disabled */
```

Definición en la línea 137 del archivo stm32f1xx_hal_conf.h.

9.3.2.58. USE_HAL_CEC_REGISTER_CALLBACKS

```
#define USE_HAL_CEC_REGISTER_CALLBACKS 0U /* CEC register callback disabled */
```

Definición en la línea 138 del archivo stm32f1xx_hal_conf.h.

9.3.2.59. USE_HAL_DAC_REGISTER_CALLBACKS

```
#define USE_HAL_DAC_REGISTER_CALLBACKS 0U /* DAC register callback disabled */
```

Definición en la línea 139 del archivo stm32f1xx_hal_conf.h.

9.3.2.60. USE_HAL_ETH_REGISTER_CALLBACKS

```
#define USE_HAL_ETH_REGISTER_CALLBACKS 0U /* ETH register callback disabled */
```

Definición en la línea 140 del archivo stm32f1xx_hal_conf.h.

9.3.2.61. USE_HAL_HCD_REGISTER_CALLBACKS

```
#define USE_HAL_HCD_REGISTER_CALLBACKS 0U /* HCD register callback disabled */
```

Definición en la línea 141 del archivo stm32f1xx_hal_conf.h.

9.3.2.62. USE_HAL_I2C_REGISTER_CALLBACKS

```
#define USE_HAL_I2C_REGISTER_CALLBACKS 0U /* I2C register callback disabled */
```

Definición en la línea 142 del archivo stm32f1xx_hal_conf.h.

9.3.2.63. USE_HAL_I2S_REGISTER_CALLBACKS

```
#define USE_HAL_I2S_REGISTER_CALLBACKS 0U /* I2S register callback disabled */
```

Definición en la línea 143 del archivo stm32f1xx_hal_conf.h.

9.3.2.64. USE_HAL_IRDA_REGISTER_CALLBACKS

```
#define USE_HAL_IRDA_REGISTER_CALLBACKS 0U /* IRDA register callback disabled */
```

Definición en la línea 152 del archivo stm32f1xx_hal_conf.h.

9.3.2.65. USE_HAL_MMC_REGISTER_CALLBACKS

```
#define USE_HAL_MMC_REGISTER_CALLBACKS 0U /* MMC register callback disabled */
```

Definición en la línea 144 del archivo stm32f1xx_hal_conf.h.

9.3.2.66. USE_HAL_NAND_REGISTER_CALLBACKS

```
#define USE_HAL_NAND_REGISTER_CALLBACKS 0U /* NAND register callback disabled */
```

Definición en la línea 145 del archivo stm32f1xx_hal_conf.h.

9.3.2.67. USE_HAL_NOR_REGISTER_CALLBACKS

```
#define USE_HAL_NOR_REGISTER_CALLBACKS 0U /* NOR register callback disabled */
```

Definición en la línea 146 del archivo stm32f1xx_hal_conf.h.

9.3.2.68. USE_HAL_PCCARD_REGISTER_CALLBACKS

```
#define USE_HAL_PCCARD_REGISTER_CALLBACKS 0U /* PCCARD register callback disabled */
```

Definición en la línea 147 del archivo stm32f1xx_hal_conf.h.

9.3.2.69. USE_HAL_PCD_REGISTER_CALLBACKS

```
#define USE_HAL_PCD_REGISTER_CALLBACKS 0U /* PCD register callback disabled */
```

Definición en la línea 148 del archivo stm32f1xx_hal_conf.h.

9.3.2.70. USE_HAL_RTC_REGISTER_CALLBACKS

```
#define USE_HAL_RTC_REGISTER_CALLBACKS 0U /* RTC register callback disabled */
```

Definición en la línea 149 del archivo stm32f1xx_hal_conf.h.

9.3.2.71. USE_HAL_SD_REGISTER_CALLBACKS

```
#define USE_HAL_SD_REGISTER_CALLBACKS 0U /* SD register callback disabled */
```

Definición en la línea 150 del archivo stm32f1xx_hal_conf.h.

9.3.2.72. USE_HAL_SMARTCARD_REGISTER_CALLBACKS

```
#define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U /* SMARTCARD register callback disabled */
```

Definición en la línea 151 del archivo stm32f1xx_hal_conf.h.

9.3.2.73. USE_HAL_SPI_REGISTER_CALLBACKS

```
#define USE_HAL_SPI_REGISTER_CALLBACKS 0U /* SPI register callback disabled */
```

Definición en la línea 154 del archivo stm32f1xx_hal_conf.h.

9.3.2.74. USE_HAL_SRAM_REGISTER_CALLBACKS

```
#define USE_HAL_SRAM_REGISTER_CALLBACKS 0U /* SRAM register callback disabled */
```

Definición en la línea 153 del archivo stm32f1xx_hal_conf.h.

9.3.2.75. USE_HAL_TIM_REGISTER_CALLBACKS

```
#define USE_HAL_TIM_REGISTER_CALLBACKS 0U /* TIM register callback disabled */
```

Definición en la línea 155 del archivo stm32f1xx_hal_conf.h.

9.3.2.76. USE_HAL_UART_REGISTER_CALLBACKS

```
#define USE_HAL_UART_REGISTER_CALLBACKS 0U /* UART register callback disabled */
```

Definición en la línea 156 del archivo stm32f1xx_hal_conf.h.

9.3.2.77. USE_HAL_USART_REGISTER_CALLBACKS

```
#define USE_HAL_USART_REGISTER_CALLBACKS 0U /* USART register callback disabled */
```

Definición en la línea 157 del archivo stm32f1xx_hal_conf.h.

9.3.2.78. USE_HAL_WWDG_REGISTER_CALLBACKS

```
#define USE_HAL_WWDG_REGISTER_CALLBACKS 0U /* WWDG register callback disabled */
```

Definición en la línea 158 del archivo stm32f1xx_hal_conf.h.

9.3.2.79. USE_RTOS

```
#define USE_RTOS 0U
```

Definición en la línea 133 del archivo stm32f1xx_hal_conf.h.

9.3.2.80. USE_SPI_CRC

```
#define USE_SPI_CRC 0U
```

Definición en la línea 230 del archivo stm32f1xx_hal_conf.h.

9.3.2.81. VDD_VALUE

```
#define VDD_VALUE 3300U
```

This is the HAL system configuration section.

Value of VDD in mv

Definición en la línea 131 del archivo stm32f1xx_hal_conf.h.

9.4. stm32f1xx_hal_conf.h

Ir a la documentación de este archivo.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __STM32F1xx_HAL_CONF_H
00022 #define __STM32F1xx_HAL_CONF_H
00023
00024 #ifdef __cplusplus
00025  extern "C" {
00026 #endif
00027
00028 /* Exported types ------------------------------------------------------------*/
00029 /* Exported constants --------------------------------------------------------*/
00030
00031 /* ########################## Module Selection ############################## */
00035
00036 #define HAL_MODULE_ENABLED
00037   /*#define HAL_ADC_MODULE_ENABLED   */
00038 /*#define HAL_CRYP_MODULE_ENABLED   */
00039 /*#define HAL_CAN_MODULE_ENABLED   */
00040 /*#define HAL_CAN_LEGACY_MODULE_ENABLED   */
00041 /*#define HAL_CEC_MODULE_ENABLED   */
00042 /*#define HAL_CORTEX_MODULE_ENABLED   */
00043 /*#define HAL_CRC_MODULE_ENABLED   */
00044 /*#define HAL_DAC_MODULE_ENABLED   */
00045 #define HAL_DMA_MODULE_ENABLED
00046 /*#define HAL_ETH_MODULE_ENABLED   */
00047 /*#define HAL_FLASH_MODULE_ENABLED   */
00048 #define HAL_GPIO_MODULE_ENABLED
00049 /*#define HAL_I2C_MODULE_ENABLED   */
00050 /*#define HAL_I2S_MODULE_ENABLED   */
00051 /*#define HAL_IRDA_MODULE_ENABLED   */
00052 /*#define HAL_IWDG_MODULE_ENABLED   */
00053 /*#define HAL_NOR_MODULE_ENABLED   */
00054 /*#define HAL_NAND_MODULE_ENABLED   */
00055 /*#define HAL_PCCARD_MODULE_ENABLED   */
00056 /*#define HAL_PCD_MODULE_ENABLED   */
00057 /*#define HAL_HCD_MODULE_ENABLED   */
00058 /*#define HAL_PWR_MODULE_ENABLED   */
00059 /*#define HAL_RCC_MODULE_ENABLED   */
00060 /*#define HAL_RTC_MODULE_ENABLED   */
00061 /*#define HAL_SD_MODULE_ENABLED   */
00062 /*#define HAL_MMC_MODULE_ENABLED   */
00063 /*#define HAL_SDRAM_MODULE_ENABLED   */
00064 /*#define HAL_SMARTCARD_MODULE_ENABLED   */
00065 #define HAL_SPI_MODULE_ENABLED
00066 /*#define HAL_SRAM_MODULE_ENABLED   */
00067 #define HAL_TIM_MODULE_ENABLED
00068 #define HAL_UART_MODULE_ENABLED
00069 /*#define HAL_USART_MODULE_ENABLED   */
00070 /*#define HAL_WWDG_MODULE_ENABLED   */
```

```
00071
00072 #define HAL_CORTEX_MODULE_ENABLED
00073 #define HAL_DMA_MODULE_ENABLED
00074 #define HAL_FLASH_MODULE_ENABLED
00075 #define HAL_EXTI_MODULE_ENABLED
00076 #define HAL_GPIO_MODULE_ENABLED
00077 #define HAL_PWR_MODULE_ENABLED
00078 #define HAL_RCC_MODULE_ENABLED
00079
00080 /* ########################## Oscillator Values adaptation ####################*/
00086 #if !defined  (HSE_VALUE)
00087   #define HSE_VALUE    8000000U
00088 #endif /* HSE_VALUE */
00089
00090 #if !defined  (HSE_STARTUP_TIMEOUT)
00091   #define HSE_STARTUP_TIMEOUT    100U
00092 #endif /* HSE_STARTUP_TIMEOUT */
00093
00099 #if !defined  (HSI_VALUE)
00100   #define HSI_VALUE    8000000U
00101 #endif /* HSI_VALUE */
00102
00106 #if !defined  (LSI_VALUE)
00107  #define LSI_VALUE               40000U
00108 #endif /* LSI_VALUE */
00111
00116 #if !defined  (LSE_VALUE)
00117   #define LSE_VALUE    32768U
00118 #endif /* LSE_VALUE */
00119
00120 #if !defined  (LSE_STARTUP_TIMEOUT)
00121   #define LSE_STARTUP_TIMEOUT    5000U
00122 #endif /* LSE_STARTUP_TIMEOUT */
00123
00124 /* Tip: To avoid modifying this file each time you need to use different HSE,
00125    ===  you can define the HSE value in your toolchain compiler preprocessor. */
00126
00127 /* ########################### System Configuration ######################### */
00131 #define  VDD_VALUE                    3300U
00132 #define  TICK_INT_PRIORITY            15U
00133 #define  USE_RTOS                     0U
00134 #define  PREFETCH_ENABLE              1U
00135
00136 #define  USE_HAL_ADC_REGISTER_CALLBACKS         0U /* ADC register callback disabled       */
00137 #define  USE_HAL_CAN_REGISTER_CALLBACKS         0U /* CAN register callback disabled       */
00138 #define  USE_HAL_CEC_REGISTER_CALLBACKS         0U /* CEC register callback disabled       */
00139 #define  USE_HAL_DAC_REGISTER_CALLBACKS         0U /* DAC register callback disabled       */
00140 #define  USE_HAL_ETH_REGISTER_CALLBACKS         0U /* ETH register callback disabled       */
00141 #define  USE_HAL_HCD_REGISTER_CALLBACKS         0U /* HCD register callback disabled       */
00142 #define  USE_HAL_I2C_REGISTER_CALLBACKS         0U /* I2C register callback disabled       */
00143 #define  USE_HAL_I2S_REGISTER_CALLBACKS         0U /* I2S register callback disabled       */
00144 #define  USE_HAL_MMC_REGISTER_CALLBACKS         0U /* MMC register callback disabled       */
00145 #define  USE_HAL_NAND_REGISTER_CALLBACKS        0U /* NAND register callback disabled      */
00146 #define  USE_HAL_NOR_REGISTER_CALLBACKS         0U /* NOR register callback disabled       */
00147 #define  USE_HAL_PCCARD_REGISTER_CALLBACKS      0U /* PCCARD register callback disabled    */
00148 #define  USE_HAL_PCD_REGISTER_CALLBACKS         0U /* PCD register callback disabled       */
00149 #define  USE_HAL_RTC_REGISTER_CALLBACKS         0U /* RTC register callback disabled       */
00150 #define  USE_HAL_SD_REGISTER_CALLBACKS          0U /* SD register callback disabled        */
00151 #define  USE_HAL_SMARTCARD_REGISTER_CALLBACKS   0U /* SMARTCARD register callback disabled */
00152 #define  USE_HAL_IRDA_REGISTER_CALLBACKS        0U /* IRDA register callback disabled      */
00153 #define  USE_HAL_SRAM_REGISTER_CALLBACKS        0U /* SRAM register callback disabled      */
00154 #define  USE_HAL_SPI_REGISTER_CALLBACKS         0U /* SPI register callback disabled       */
00155 #define  USE_HAL_TIM_REGISTER_CALLBACKS         0U /* TIM register callback disabled       */
00156 #define  USE_HAL_UART_REGISTER_CALLBACKS        0U /* UART register callback disabled      */
00157 #define  USE_HAL_USART_REGISTER_CALLBACKS       0U /* USART register callback disabled     */
00158 #define  USE_HAL_WWDG_REGISTER_CALLBACKS        0U /* WWDG register callback disabled      */
00159
00160 /* ########################## Assert Selection ############################## */
00165 /* #define USE_FULL_ASSERT    1U */
00166
00167 /* ################### Ethernet peripheral configuration ##################### */
00168
00169 /* Section 1 : Ethernet peripheral configuration */
00170
00171 /* MAC ADDRESS: MAC_ADDR0:MAC_ADDR1:MAC_ADDR2:MAC_ADDR3:MAC_ADDR4:MAC_ADDR5 */
00172 #define MAC_ADDR0   2U
00173 #define MAC_ADDR1   0U
00174 #define MAC_ADDR2   0U
00175 #define MAC_ADDR3   0U
00176 #define MAC_ADDR4   0U
00177 #define MAC_ADDR5   0U
00178
00179 /* Definition of the Ethernet driver buffers size and count */
00180 #define ETH_RX_BUF_SIZE                ETH_MAX_PACKET_SIZE /* buffer size for receive               */
00181 #define ETH_TX_BUF_SIZE                ETH_MAX_PACKET_SIZE /* buffer size for transmit              */
00182 #define ETH_RXBUFNB                    8U       /* 4 Rx buffers of size ETH_RX_BUF_SIZE  */
00183 #define ETH_TXBUFNB                    4U       /* 4 Tx buffers of size ETH_TX_BUF_SIZE  */
```

```
00184
00185 /* Section 2: PHY configuration section */
00186
00187 /* DP83848_PHY_ADDRESS Address*/
00188 #define DP83848_PHY_ADDRESS           0x01U
00189 /* PHY Reset delay these values are based on a 1 ms Systick interrupt*/
00190 #define PHY_RESET_DELAY                 0x000000FFU
00191 /* PHY Configuration delay */
00192 #define PHY_CONFIG_DELAY                0x00000FFFU
00193
00194 #define PHY_READ_TO                     0x0000FFFFU
00195 #define PHY_WRITE_TO                    0x0000FFFFU
00196
00197 /* Section 3: Common PHY Registers */
00198
00199 #define PHY_BCR                         ((uint16_t)0x00)
00200 #define PHY_BSR                         ((uint16_t)0x01)
00201
00202 #define PHY_RESET                       ((uint16_t)0x8000)
00203 #define PHY_LOOPBACK                    ((uint16_t)0x4000)
00204 #define PHY_FULLDUPLEX_100M             ((uint16_t)0x2100)
00205 #define PHY_HALFDUPLEX_100M             ((uint16_t)0x2000)
00206 #define PHY_FULLDUPLEX_10M              ((uint16_t)0x0100)
00207 #define PHY_HALFDUPLEX_10M              ((uint16_t)0x0000)
00208 #define PHY_AUTONEGOTIATION             ((uint16_t)0x1000)
00209 #define PHY_RESTART_AUTONEGOTIATION     ((uint16_t)0x0200)
00210 #define PHY_POWERDOWN                   ((uint16_t)0x0800)
00211 #define PHY_ISOLATE                     ((uint16_t)0x0400)
00212
00213 #define PHY_AUTONEGO_COMPLETE           ((uint16_t)0x0020)
00214 #define PHY_LINKED_STATUS               ((uint16_t)0x0004)
00215 #define PHY_JABBER_DETECTION            ((uint16_t)0x0002)
00216
00217 /* Section 4: Extended PHY Registers */
00218 #define PHY_SR                          ((uint16_t)0x10U)
00219
00220 #define PHY_SPEED_STATUS                ((uint16_t)0x0002U)
00221 #define PHY_DUPLEX_STATUS               ((uint16_t)0x0004U)
00222
00223 /* ################## SPI peripheral configuration ########################### */
00224
00225 /* CRC FEATURE: Use to activate CRC feature inside HAL SPI Driver
00226 * Activated: CRC code is present inside driver
00227 * Deactivated: CRC code cleaned from driver
00228 */
00229
00230 #define USE_SPI_CRC                     0U
00231
00232 /* Includes ------------------------------------------------------------------*/
00236
00237 #ifdef HAL_RCC_MODULE_ENABLED
00238 #include "stm32f1xx_hal_rcc.h"
00239 #endif /* HAL_RCC_MODULE_ENABLED */
00240
00241 #ifdef HAL_GPIO_MODULE_ENABLED
00242 #include "stm32f1xx_hal_gpio.h"
00243 #endif /* HAL_GPIO_MODULE_ENABLED */
00244
00245 #ifdef HAL_EXTI_MODULE_ENABLED
00246 #include "stm32f1xx_hal_exti.h"
00247 #endif /* HAL_EXTI_MODULE_ENABLED */
00248
00249 #ifdef HAL_DMA_MODULE_ENABLED
00250 #include "stm32f1xx_hal_dma.h"
00251 #endif /* HAL_DMA_MODULE_ENABLED */
00252
00253 #ifdef HAL_ETH_MODULE_ENABLED
00254 #include "stm32f1xx_hal_eth.h"
00255 #endif /* HAL_ETH_MODULE_ENABLED */
00256
00257 #ifdef HAL_CAN_MODULE_ENABLED
00258 #include "stm32f1xx_hal_can.h"
00259 #endif /* HAL_CAN_MODULE_ENABLED */
00260
00261 #ifdef HAL_CAN_LEGACY_MODULE_ENABLED
00262   #include "Legacy/stm32f1xx_hal_can_legacy.h"
00263 #endif /* HAL_CAN_LEGACY_MODULE_ENABLED */
00264
00265 #ifdef HAL_CEC_MODULE_ENABLED
00266 #include "stm32f1xx_hal_cec.h"
00267 #endif /* HAL_CEC_MODULE_ENABLED */
00268
00269 #ifdef HAL_CORTEX_MODULE_ENABLED
00270 #include "stm32f1xx_hal_cortex.h"
00271 #endif /* HAL_CORTEX_MODULE_ENABLED */
00272
00273 #ifdef HAL_ADC_MODULE_ENABLED
```

```
00274 #include "stm32f1xx_hal_adc.h"
00275 #endif /* HAL_ADC_MODULE_ENABLED */
00276
00277 #ifdef HAL_CRC_MODULE_ENABLED
00278 #include "stm32f1xx_hal_crc.h"
00279 #endif /* HAL_CRC_MODULE_ENABLED */
00280
00281 #ifdef HAL_DAC_MODULE_ENABLED
00282 #include "stm32f1xx_hal_dac.h"
00283 #endif /* HAL_DAC_MODULE_ENABLED */
00284
00285 #ifdef HAL_FLASH_MODULE_ENABLED
00286 #include "stm32f1xx_hal_flash.h"
00287 #endif /* HAL_FLASH_MODULE_ENABLED */
00288
00289 #ifdef HAL_SRAM_MODULE_ENABLED
00290 #include "stm32f1xx_hal_sram.h"
00291 #endif /* HAL_SRAM_MODULE_ENABLED */
00292
00293 #ifdef HAL_NOR_MODULE_ENABLED
00294 #include "stm32f1xx_hal_nor.h"
00295 #endif /* HAL_NOR_MODULE_ENABLED */
00296
00297 #ifdef HAL_I2C_MODULE_ENABLED
00298 #include "stm32f1xx_hal_i2c.h"
00299 #endif /* HAL_I2C_MODULE_ENABLED */
00300
00301 #ifdef HAL_I2S_MODULE_ENABLED
00302 #include "stm32f1xx_hal_i2s.h"
00303 #endif /* HAL_I2S_MODULE_ENABLED */
00304
00305 #ifdef HAL_IWDG_MODULE_ENABLED
00306 #include "stm32f1xx_hal_iwdg.h"
00307 #endif /* HAL_IWDG_MODULE_ENABLED */
00308
00309 #ifdef HAL_PWR_MODULE_ENABLED
00310 #include "stm32f1xx_hal_pwr.h"
00311 #endif /* HAL_PWR_MODULE_ENABLED */
00312
00313 #ifdef HAL_RTC_MODULE_ENABLED
00314 #include "stm32f1xx_hal_rtc.h"
00315 #endif /* HAL_RTC_MODULE_ENABLED */
00316
00317 #ifdef HAL_PCCARD_MODULE_ENABLED
00318 #include "stm32f1xx_hal_pccard.h"
00319 #endif /* HAL_PCCARD_MODULE_ENABLED */
00320
00321 #ifdef HAL_SD_MODULE_ENABLED
00322 #include "stm32f1xx_hal_sd.h"
00323 #endif /* HAL_SD_MODULE_ENABLED */
00324
00325 #ifdef HAL_NAND_MODULE_ENABLED
00326 #include "stm32f1xx_hal_nand.h"
00327 #endif /* HAL_NAND_MODULE_ENABLED */
00328
00329 #ifdef HAL_SPI_MODULE_ENABLED
00330 #include "stm32f1xx_hal_spi.h"
00331 #endif /* HAL_SPI_MODULE_ENABLED */
00332
00333 #ifdef HAL_TIM_MODULE_ENABLED
00334 #include "stm32f1xx_hal_tim.h"
00335 #endif /* HAL_TIM_MODULE_ENABLED */
00336
00337 #ifdef HAL_UART_MODULE_ENABLED
00338 #include "stm32f1xx_hal_uart.h"
00339 #endif /* HAL_UART_MODULE_ENABLED */
00340
00341 #ifdef HAL_USART_MODULE_ENABLED
00342 #include "stm32f1xx_hal_usart.h"
00343 #endif /* HAL_USART_MODULE_ENABLED */
00344
00345 #ifdef HAL_IRDA_MODULE_ENABLED
00346 #include "stm32f1xx_hal_irda.h"
00347 #endif /* HAL_IRDA_MODULE_ENABLED */
00348
00349 #ifdef HAL_SMARTCARD_MODULE_ENABLED
00350 #include "stm32f1xx_hal_smartcard.h"
00351 #endif /* HAL_SMARTCARD_MODULE_ENABLED */
00352
00353 #ifdef HAL_WWDG_MODULE_ENABLED
00354 #include "stm32f1xx_hal_wwdg.h"
00355 #endif /* HAL_WWDG_MODULE_ENABLED */
00356
00357 #ifdef HAL_PCD_MODULE_ENABLED
00358 #include "stm32f1xx_hal_pcd.h"
00359 #endif /* HAL_PCD_MODULE_ENABLED */
00360
```

```
00361 #ifdef HAL_HCD_MODULE_ENABLED
00362 #include "stm32f1xx_hal_hcd.h"
00363 #endif /* HAL_HCD_MODULE_ENABLED */
00364
00365 #ifdef HAL_MMC_MODULE_ENABLED
00366 #include "stm32f1xx_hal_mmc.h"
00367 #endif /* HAL_MMC_MODULE_ENABLED */
00368
00369 /* Exported macro ------------------------------------------------------------*/
00370 #ifdef  USE_FULL_ASSERT
00379 #define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))
00380 /* Exported functions --------------------------------------------------------- */
00381 void assert_failed(uint8_t* file, uint32_t line);
00382 #else
00383 #define assert_param(expr) ((void)0U)
00384 #endif /* USE_FULL_ASSERT */
00385
00386 #ifdef __cplusplus
00387 }
00388 #endif
00389
00390 #endif /* __STM32F1xx_HAL_CONF_H */
00391
```

9.5. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Inc/stm32f1xx_it.h

This file contains the headers of the interrupt handlers.

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Funciones

- void NMI_Handler (void)

  This function handles Non maskable interrupt.
- void HardFault_Handler (void)

  This function handles Hard fault interrupt.
- void MemManage_Handler (void)

  This function handles Memory management fault.
- void BusFault_Handler (void)

  This function handles Prefetch fault, memory access fault.
- void UsageFault_Handler (void)

  This function handles Undefined instruction or illegal state.
- void SVC_Handler (void)

  This function handles System service call via SWI instruction.
- void DebugMon_Handler (void)

  This function handles Debug monitor.
- void PendSV_Handler (void)

  This function handles Pendable request for system service.
- void SysTick_Handler (void)

  This function handles System tick timer.
- void DMA1_Channel4_IRQHandler (void)

  This function handles DMA1 channel4 global interrupt.
- void DMA1_Channel5_IRQHandler (void)

  This function handles DMA1 channel5 global interrupt.
- void TIM2_IRQHandler (void)

This function handles TIM2 global interrupt.

- void TIM3_IRQHandler (void)

  This function handles TIM3 global interrupt.

- void SPI2_IRQHandler (void)

  This function handles SPI2 global interrupt.

9.5.1. Descripción detallada

This file contains the headers of the interrupt handlers.

Atención

Definición en el archivo stm32f1xx_it.h.

9.5.2. Documentación de funciones

9.5.2.1. BusFault_Handler()

```
void BusFault_Handler (
             void )
```

This function handles Prefetch fault, memory access fault.

Definición en la línea 62 del archivo stm32f1xx_it.c.

9.5.2.2. DebugMon_Handler()

```
void DebugMon_Handler (
             void )
```

This function handles Debug monitor.

Definición en la línea 89 del archivo stm32f1xx_it.c.

9.5.2.3. DMA1_Channel4_IRQHandler()

```
void DMA1_Channel4_IRQHandler (
             void )
```

This function handles DMA1 channel4 global interrupt.

Definición en la línea 110 del archivo stm32f1xx_it.c.

9.5.2.4. DMA1_Channel5_IRQHandler()

```
void DMA1_Channel5_IRQHandler (
            void )
```

This function handles DMA1 channel5 global interrupt.

Definición en la línea 117 del archivo stm32f1xx_it.c.

9.5.2.5. HardFault_Handler()

```
void HardFault_Handler (
            void )
```

This function handles Hard fault interrupt.

Definición en la línea 42 del archivo stm32f1xx_it.c.

9.5.2.6. MemManage_Handler()

```
void MemManage_Handler (
            void )
```

This function handles Memory management fault.

Definición en la línea 52 del archivo stm32f1xx_it.c.

9.5.2.7. NMI_Handler()

```
void NMI_Handler (
            void )
```

This function handles Non maskable interrupt.

Definición en la línea 32 del archivo stm32f1xx_it.c.

9.5.2.8. PendSV_Handler()

```
void PendSV_Handler (
            void )
```

This function handles Pendable request for system service.

Definición en la línea 96 del archivo stm32f1xx_it.c.

9.5.2.9. SPI2_IRQHandler()

```
void SPI2_IRQHandler (
            void )
```

This function handles SPI2 global interrupt.

Definición en la línea 139 del archivo stm32f1xx_it.c.

9.5.2.10. SVC_Handler()

```
void SVC_Handler (
             void )
```

This function handles System service call via SWI instruction.

Definición en la línea 82 del archivo stm32f1xx_it.c.

9.5.2.11. SysTick_Handler()

```
void SysTick_Handler (
             void )
```

This function handles System tick timer.

Definición en la línea 103 del archivo stm32f1xx_it.c.

9.5.2.12. TIM2_IRQHandler()

```
void TIM2_IRQHandler (
             void )
```

This function handles TIM2 global interrupt.

Definición en la línea 124 del archivo stm32f1xx_it.c.

Gráfico de llamadas de esta función:

9.5.2.13. TIM3_IRQHandler()

```
void TIM3_IRQHandler (
             void )
```

This function handles TIM3 global interrupt.

Definición en la línea 132 del archivo stm32f1xx_it.c.

9.5.2.14. UsageFault_Handler()

```
void UsageFault_Handler (
             void )
```

This function handles Undefined instruction or illegal state.

Definición en la línea 72 del archivo stm32f1xx_it.c.

9.6. stm32f1xx_it.h

Ir a la documentación de este archivo.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __STM32F1xx_IT_H
00022 #define __STM32F1xx_IT_H
00023
00024 #ifdef __cplusplus
00025  extern "C" {
00026 #endif
00027
00028 /* Private includes ----------------------------------------------------------*/
00029 /* USER CODE BEGIN Includes */
00030
00031 /* USER CODE END Includes */
00032
00033 /* Exported types ------------------------------------------------------------*/
00034 /* USER CODE BEGIN ET */
00035
00036 /* USER CODE END ET */
00037
00038 /* Exported constants --------------------------------------------------------*/
00039 /* USER CODE BEGIN EC */
00040
00041 /* USER CODE END EC */
00042
00043 /* Exported macro ------------------------------------------------------------*/
00044 /* USER CODE BEGIN EM */
00045
00046 /* USER CODE END EM */
00047
00048 /* Exported functions prototypes ---------------------------------------------*/
00049 void NMI_Handler(void);
00050 void HardFault_Handler(void);
00051 void MemManage_Handler(void);
00052 void BusFault_Handler(void);
00053 void UsageFault_Handler(void);
00054 void SVC_Handler(void);
00055 void DebugMon_Handler(void);
00056 void PendSV_Handler(void);
00057 void SysTick_Handler(void);
00058 void DMA1_Channel4_IRQHandler(void);
00059 void DMA1_Channel5_IRQHandler(void);
00060 void TIM2_IRQHandler(void);
00061 void TIM3_IRQHandler(void);
00062 void SPI2_IRQHandler(void);
00063 /* USER CODE BEGIN EFP */
00064
00065 /* USER CODE END EFP */
00066
00067 #ifdef __cplusplus
00068 }
00069 #endif
00070
00071 #endif /* __STM32F1xx_IT_H */
```

9.7. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_Estados/GestorEstados.c

Implementa la máquina de estados del LVFV.

```
#include "GestorEstados.h"
#include "../Gestor_SVM/GestorSVM.h"
```

Gráfico de dependencias incluidas en GestorEstados.c:

9.8. GestorEstados.c

Ir a la documentación de este archivo.

```
00001
00006
00007 #include "GestorEstados.h"
00008 #include "../Gestor_SVM/GestorSVM.h"
00009
00010 static SystemState currentState = STATE_INIT;
00011
00012 SystemActionResponse GestorEstados_Action(SystemAction sysAct, int value) {
00013     SystemActionResponse retVal = ACTION_RESP_ERR;  // default seguro
00014     switch(sysAct) {
00015         case ACTION_INIT_DONE:
00016             if(currentState == STATE_INIT) {
00017                 currentState = STATE_IDLE;
00018                 retVal = ACTION_RESP_OK;
00019             } else {
00020                 retVal = ACTION_RESP_ERR;
00021             }
00022             break;
00023         case ACTION_TO_IDLE:
00024             if(currentState == STATE_BRAKING) {
00025                 currentState = STATE_IDLE;
00026                 retVal = ACTION_RESP_OK;
00027             } else {
00028                 retVal = ACTION_RESP_ERR;
00029             }
00030             break;
00031         case ACTION_START:
00032             if(currentState == STATE_IDLE) {
00033                 GestorSVM_MotorStart();
00034                 currentState = STATE_VEL_CHANGE;
00035                 retVal = ACTION_RESP_OK;
00036             }else if(currentState == STATE_RUNNING || currentState == STATE_VEL_CHANGE || currentState
      == STATE_BRAKING) {
00037                 retVal = ACTION_RESP_MOVING;
00038             }else if(currentState == STATE_EMERGENCY) {
00039                 retVal = ACTION_RESP_EMERGENCY_ACTIVE;
00040             } else {
00041                 retVal = ACTION_RESP_ERR;
00042             }
00043             break;
00044         case ACTION_STOP:
00045             if(currentState == STATE_RUNNING || currentState == STATE_VEL_CHANGE) {
00046                 GestorSVM_MotorStop();
00048                 currentState = STATE_BRAKING;
00049                 retVal = ACTION_RESP_OK;
00050             } else if(currentState == STATE_EMERGENCY) {
00051                 GestorSVM_Estop();
00052                 currentState = STATE_IDLE;
00053                 retVal = ACTION_RESP_OK;
00054             } else if (currentState == STATE_IDLE) {
00055                 retVal = ACTION_RESP_NOT_MOVING;
00056             } else {
00057                 retVal = ACTION_RESP_ERR;
00058             }
00059             break;
00060         case ACTION_MOTOR_STOPPED:
00061             if(currentState == STATE_BRAKING) {
00062                 currentState = STATE_IDLE;
00063                 retVal = ACTION_RESP_OK;
00064             } else {
00065                 retVal = ACTION_RESP_ERR;
00066             }
00067             break;
00068         case ACTION_EMERGENCY:
00069             if(currentState != STATE_EMERGENCY) {
00071                 GestorSVM_Estop();
00072                 currentState = STATE_EMERGENCY;
00073                 retVal = ACTION_RESP_OK;
00074             } else {
00075                 retVal = ACTION_RESP_ERR;
00076             }
00077             break;
00078         case ACTION_TO_CONST_RUNNING:
00079             if(currentState == STATE_VEL_CHANGE) {
00080                 currentState = STATE_RUNNING;
00081                 retVal = ACTION_RESP_OK;
00082             } else {
00083                 retVal = ACTION_RESP_ERR;
00084             }
00085             break;
00086         case ACTION_SET_FREC:
```

```
00087             if(currentState == STATE_IDLE || currentState == STATE_RUNNING || currentState ==
      STATE_VEL_CHANGE) {
00088                 int conf_retVal = GestorSVM_SetFrec(value);
00089                 if(conf_retVal == 0 || conf_retVal == -2) {
00090                     retVal = ACTION_RESP_OK;
00091                 } else if(conf_retVal == 1) {
00092                     currentState = STATE_VEL_CHANGE;
00093                     retVal = ACTION_RESP_OK;
00094                 } else if(conf_retVal == -1) {
00095                     retVal = ACTION_RESP_OUT_RANGE;
00096                 }
00097             } else if( currentState == STATE_BRAKING) {
00098                 retVal = ACTION_RESP_MOVING;
00099             } else {
00100                 retVal = ACTION_RESP_ERR;
00101             }
00102             break;
00103         case ACTION_SET_ACEL:
00104             // Implementar la logica para cambiar la aceleracion
00105             if(currentState == STATE_IDLE || currentState == STATE_RUNNING) {
00106                 switch( GestorSVM_SetAcel(value) ) {
00107                     case 0:
00108                         retVal = ACTION_RESP_OK;
00109                         break;
00110                     case -1:
00111                         retVal = ACTION_RESP_OUT_RANGE;
00112                         break;
00113                     case -2:
00114                     default:
00115                         retVal = ACTION_RESP_ERR;
00116                         break;
00117                 }
00118             } else {
00119                 retVal = ACTION_RESP_ERR;
00120             }
00121             break;
00122         case ACTION_SET_DESACEL:
00123             if(currentState == STATE_IDLE || currentState == STATE_RUNNING) {
00124                 switch( GestorSVM_SetDecel(value) ) {
00125                     case 0:
00126                         retVal = ACTION_RESP_OK;
00127                         break;
00128                     case -1:
00129                         retVal = ACTION_RESP_OUT_RANGE;
00130                         break;
00131                     case -2:
00132                     default:
00133                         retVal = ACTION_RESP_ERR;
00134                         break;
00135                 }
00136             } else {
00137                 retVal = ACTION_RESP_ERR;
00138             }
00139             break;
00140         case ACTION_SET_DIR:
00141             if(currentState == STATE_IDLE) {
00142                 switch( GestorSVM_SetDir(value)) {
00143                     case 0:
00144                         retVal = ACTION_RESP_OK;
00145                         break;
00146                     case -1:
00147                         retVal = ACTION_RESP_OUT_RANGE;
00148                         break;
00149                     case -2:
00150                         retVal = ACTION_RESP_MOVING;
00151                         break;
00152                     default:
00153                         retVal = ACTION_RESP_ERR;
00154                         break;
00155                 }
00156             } else {
00157                 retVal = ACTION_RESP_MOVING;
00158             }
00159             break;
00160         // case ACTION_GET_FREC:
00161         // case ACTION_GET_ACEL:
00162         // case ACTION_GET_DESACEL:
00163         // case ACTION_GET_DIR:
00164         //     retVal = ACTION_RESP_OK;     // lectura válida, sin transición
00165         //     break;
00166         case ACTION_IS_MOTOR_STOP:
00167             if(currentState == STATE_IDLE || currentState == STATE_EMERGENCY) {
00168                 retVal = ACTION_RESP_NOT_MOVING;
00169             }else {
00170                 retVal = ACTION_RESP_MOVING;
00171             }
00172             break;
```

```
00173         default:
00174             retVal = ACTION_RESP_ERR;
00175             break;
00176     }
00177     return retVal;
00178 }
```

9.9. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_Estados/GestorEstados.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Enumeraciones

- enum SystemState {
  STATE_INIT = 0 , STATE_IDLE , STATE_RUNNING , STATE_VEL_CHANGE ,
  STATE_BRAKING , STATE_EMERGENCY }

  Estados de la máquina de estados del variador.
- enum SystemAction {
  ACTION_NONE = 0 , ACTION_INIT_DONE = 1 , ACTION_TO_IDLE , ACTION_START ,
  ACTION_STOP , ACTION_MOTOR_STOPPED , ACTION_EMERGENCY , ACTION_SET_FREC ,
  ACTION_SET_ACEL , ACTION_SET_DESACEL , ACTION_SET_DIR , ACTION_TO_CONST_RUNNING ,
  ACTION_GET_FREC , ACTION_GET_ACEL , ACTION_GET_DESACEL , ACTION_GET_DIR ,
  ACTION_IS_MOTOR_STOP }

  Acciones externas/internas que gobiernan la máquina de estados del VFD.
- enum SystemActionResponse {
  ACTION_RESP_OK = 0 , ACTION_RESP_ERR = 1 , ACTION_RESP_MOVING = 2 , ACTION_RESP_NOT_MOVING = 3 ,
  ACTION_RESP_OUT_RANGE = 4 , ACTION_RESP_EMERGENCY_ACTIVE = 5 }

  Códigos de respuesta del gestor de estados ante una acción recibida.

Funciones

- SystemActionResponse GestorEstados_Action (SystemAction sysAct, int value)

  Ejecuta una acción de la máquina de estados y devuelve el resultado.

9.9.1. Documentación de enumeraciones

9.9.1.1. SystemAction

```
enum SystemAction
```

Acciones externas/internas que gobiernan la máquina de estados del VFD.

Cada acción puede provocar un cambio de estado (SystemState) y devuelve un SystemActionResponse según las reglas implementadas en GestorEstados_Action(). Las referencias a estados (STATE_∗) y respuestas (ACTION_RESP_∗) indican el flujo esperado y los códigos de retorno típicos.

Valores de enumeraciones

ACTION_NONE	Sin uso. No debe enviarse.
ACTION_INIT_DONE	Fin de inicialización. Válida **solo** si `currentState==@ref STATE_INIT`: pasa a STATE_IDLE y responde ACTION_RESP_OK; en cualquier otro estado responde ACTION_RESP_ERR.
ACTION_TO_IDLE	Forzar estado inactivo. Válida cuando `currentState==@ref STATE_BRAKING`: pasa a STATE_IDLE y responde ACTION_RESP_OK; si no está frenando, responde ACTION_RESP_ERR.
ACTION_START	Solicitud de arranque. Si `currentState==@ref STATE_IDLE` llama a `GestorSVM_MotorStart()`, pasa a STATE_VEL_CHANGE y responde ACTION_RESP_OK. Si está en STATE_RUNNING, STATE_VEL_CHANGE o STATE_BRAKING responde ACTION_RESP_MOVING. En STATE_EMERGENCY responde ACTION_RESP_EMERGENCY_ACTIVE. En cualquier otro caso ACTION_RESP_ERR.
ACTION_STOP	Solicitud de parada. Si está en STATE_RUNNING o STATE_VEL_CHANGE, llama a `GestorSVM_MotorStop()`, pasa a STATE_BRAKING y responde ACTION_RESP_OK. Si está en STATE_EMERGENCY, ejecuta `GestorSVM_Estop()`, pasa a STATE_IDLE y responde ACTION_RESP_OK. Si ya está en STATE_IDLE responde ACTION_RESP_NOT_MOVING. Otro caso: ACTION_RESP_ERR.
ACTION_MOTOR_STOPPED	Confirmación de motor detenido. Válida durante STATE_BRAKING: pasa a STATE_IDLE y responde ACTION_RESP_OK; si no está frenando responde ACTION_RESP_ERR.
ACTION_EMERGENCY	Activación de emergencia. Si no está en STATE_EMERGENCY, ejecuta `GestorSVM_Estop()`, pasa a STATE_EMERGENCY y responde ACTION_RESP_OK; si ya estaba en emergencia responde ACTION_RESP_ERR.
ACTION_SET_FREC	Fijar frecuencia de régimen. Permitido en STATE_IDLE, STATE_RUNNING o STATE_VEL_CHANGE. Llama a `GestorSVM_SetFrec(value)`: • 0 o -2 → ACTION_RESP_OK (permanece en el estado actual) • 1 → pasa a STATE_VEL_CHANGE y ACTION_RESP_OK • -1 → ACTION_RESP_OUT_RANGE Si está en STATE_BRAKING → ACTION_RESP_MOVING; otro estado → ACTION_RESP_ERR.
ACTION_SET_ACEL	Configurar aceleración. Permitido en STATE_IDLE o STATE_RUNNING. Llama a `GestorSVM_SetAcel(value)`: • 0 → ACTION_RESP_OK • -1 → ACTION_RESP_OUT_RANGE • -2 u otro → ACTION_RESP_ERR En otros estados → ACTION_RESP_ERR.
ACTION_SET_DESACEL	Configurar desaceleración. Igual a ACTION_SET_ACEL pero llamando `GestorSVM_SetDecel(value)`. Respuestas: ACTION_RESP_OK, ACTION_RESP_OUT_RANGE o ACTION_RESP_ERR. En otros estados → ACTION_RESP_ERR.

ACTION_SET_DIR	Configurar dirección de giro. Permitido solo en STATE_IDLE. Llama a `GestorSVM_SetDir(value)`: • 0 → ACTION_RESP_OK • -1 → ACTION_RESP_OUT_RANGE • -2 → ACTION_RESP_MOVING • otro → ACTION_RESP_ERR Si no está en IDLE → ACTION_RESP_MOVING.
ACTION_TO_CONST_RUNNING	Transición a régimen constante. Válida cuando `currentState==@ref STATE_VEL_CHANGE`: pasa a STATE_RUNNING y responde ACTION_RESP_OK; caso contrario ACTION_RESP_ERR.
ACTION_GET_FREC	Consulta de frecuencia actual. Acción de lectura (sin cambio de estado). La entrega del valor se realiza por la capa que invoca a GestorEstados_Action() (no hay respuesta numérica directa en este switch).
ACTION_GET_ACEL	Consulta de aceleración actual. Acción de lectura (sin cambio de estado). La obtención del valor queda a cargo de la capa llamadora.
ACTION_GET_DESACEL	Consulta de desaceleración actual. Acción de lectura (sin cambio de estado). La obtención del valor queda a cargo de la capa llamadora.
ACTION_GET_DIR	Consulta de dirección actual. Acción de lectura (sin cambio de estado). La obtención del valor queda a cargo de la capa llamadora.
ACTION_IS_MOTOR_STOP	¿El motor está detenido? Si `currentState==@ref STATE_IDLE` o STATE_EMERGENCY → ACTION_RESP_NOT_MOVING; en otro estado → ACTION_RESP_MOVING.

Definición en la línea 64 del archivo GestorEstados.h.

9.9.1.2. SystemActionResponse

`enum SystemActionResponse`

Códigos de respuesta del gestor de estados ante una acción recibida.

Los valores indican el resultado de `GestorEstados_Action()` según la acción solicitada (SystemAction) y el estado actual (SystemState). Se utilizan también como payload de respuesta en el enlace SPI con el ESP32.

Valores de enumeraciones

ACTION_RESP_OK	
ACTION_RESP_ERR	La acción fue válida y se ejecutó correctamente. Si correspondía, la transición de estado se realizó.
ACTION_RESP_MOVING	Error genérico: secuencia inválida para el estado actual o fallo al aplicar la configuración. No hay garantía de cambio de estado.
ACTION_RESP_NOT_MOVING	El motor está en movimiento (p. ej., en STATE_RUNNING, STATE_VEL_CHANGE o STATE_BRAKING). Puede usarse como respuesta a una consulta o para rechazar acciones no permitidas mientras hay movimiento.

ACTION_RESP_OUT_RANGE	El motor está detenido (p. ej., en STATE_IDLE o STATE_EMERGENCY). Puede ser respuesta a consulta o a un STOP redundante.
ACTION_RESP_EMERGENCY_ACTIVE	El parámetro de configuración recibido (frecuencia, aceleración, etc.) está fuera de los límites admitidos por la lógica (p. ej., `GestorSVM_*`).

Definición en la línea 212 del archivo GestorEstados.h.

9.9.1.3. SystemState

`enum SystemState`

Estados de la máquina de estados del variador.

Representan el ciclo de vida del sistema y condicionan qué acciones (SystemAction) son válidas y qué respuestas (SystemActionResponse) devuelve `GestorEstados_Action()`. Las transiciones típicas son:

- STATE_INIT → (ACTION_INIT_DONE) → STATE_IDLE
- STATE_IDLE → (ACTION_START) → STATE_VEL_CHANGE
- STATE_VEL_CHANGE → (ACTION_TO_CONST_RUNNING) → STATE_RUNNING
- STATE_RUNNING → (ACTION_STOP) → STATE_BRAKING
- STATE_BRAKING → (ACTION_MOTOR_STOPPED) → STATE_IDLE
- Cualquier estado → (ACTION_EMERGENCY) → STATE_EMERGENCY

Valores de enumeraciones

STATE_INIT	
STATE_IDLE	Sistema en inicialización. Solo es válida ACTION_INIT_DONE para pasar a STATE_IDLE.
STATE_RUNNING	Reposo: motor detenido (o salida de emergencia). Admite configuración (ACTION_SET_FREC, ACTION_SET_ACEL, ACTION_SET_DESACEL, ACTION_SET_DIR) y arranque con ACTION_START.
STATE_VEL_CHANGE	Marcha a velocidad constante. Acepta cambio de parámetros permitidos y ACTION_STOP para transicionar a STATE_BRAKING.
STATE_BRAKING	Transitorio de cambio de velocidad (acel./desacel.). Al alcanzar régimen↩ : ACTION_TO_CONST_RUNNING → STATE_RUNNING. ACTION_STOP lleva a STATE_BRAKING.
STATE_EMERGENCY	Frenado en curso. Al detenerse: ACTION_MOTOR_STOPPED → STATE_IDLE. ACTION_TO_IDLE fuerza el paso a inactivo. Emergencia activa (salidas deshabilitadas). Solo acciones de recuperación/acondicionamiento; p.ej. ACTION_STOP puede llevar a STATE_IDLE según la lógica de seguridad.

Definición en la línea 25 del archivo GestorEstados.h.

9.9.2. Documentación de funciones

9.9.2.1. GestorEstados_Action()

```
SystemActionResponse GestorEstados_Action (
            SystemAction sysAct,
            int value)
```

Ejecuta una acción de la máquina de estados y devuelve el resultado.

Evalúa la acción `sysAct` contra el estado actual (SystemState) y, según corresponda, invoca rutinas del SVM (p.ej. `GestorSVM_MotorStart()`, `GestorSVM_MotorStop()`, `GestorSVM_Estop()`, `GestorSVM_SetFrec/SetAcel/SetDecel/SetDir`) y actualiza la variable interna `currentState`.
Las reglas de transición/retorno (según el código mostrado) incluyen, entre otras:

- ACTION_INIT_DONE : solo desde STATE_INIT → STATE_IDLE.
- ACTION_START : desde STATE_IDLE → STATE_VEL_CHANGE; si hay movimiento → ACTION_RESP_MOVING; en emergencia → ACTION_RESP_EMERGENCY_ACTIVE.
- ACTION_STOP : desde RUNNING/VEL_CHANGE → STATE_BRAKING; en EMERGENCY → STATE_IDLE; en IDLE → ACTION_RESP_NOT_MOVING.
- ACTION_MOTOR_STOPPED : en BRAKING → STATE_IDLE.
- ACTION_EMERGENCY : a STATE_EMERGENCY (si no lo estaba).
- ACTION_TO_CONST_RUNNING : en VEL_CHANGE → STATE_RUNNING.
- ACTION_SET_FREC/SET_ACEL/SET_DESACEL/SET_DIR : validan rango/estado y pueden dejar el estado o forzar STATE_VEL_CHANGE (según retorno de `GestorSVM_*`).

Parámetros

in	*sysAct*	Acción solicitada (SystemAction).
in	*value*	Parámetro asociado a la acción (p.ej., frecuencia, acel./desacel., dirección).

Devuelve

SystemActionResponse

- ACTION_RESP_OK : La acción fue válida y se ejecutó (posible cambio de estado).
- ACTION_RESP_ERR : Secuencia inválida para el estado actual o fallo de configuración.
- ACTION_RESP_MOVING : Rechazo/indicación de que el motor está en movimiento.
- ACTION_RESP_NOT_MOVING : Rechazo/indicación de que el motor está detenido.
- ACTION_RESP_OUT_RANGE : Parámetro `value` fuera de límites admitidos.
- ACTION_RESP_EMERGENCY_ACTIVE: Acción no permitida bajo STATE_EMERGENCY.

Nota

Las acciones de lectura (ACTION_GET_FREC, ACTION_GET_ACEL, ACTION_GET_DESACEL, ACTION_GET_DIR) no cambian de estado en este switch.

Atención

Verificar que cada `case` finalice con `break` (en el código provisto faltaba `break` tras ACTION_EMERGENCY para evitar fall-through).

Tareas pendientes : Aca tengo que poner un timer para que calcule el tiempo aprox de frenado.

Tareas pendientes : Revisar si es necesario chequear esto ¿Por qué no ejecutamos el stop sin importar el estado?

Definición en la línea 12 del archivo GestorEstados.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.10. GestorEstados.h

Ir a la documentación de este archivo.

```
00001 /*
00002  * GestorEstados.h
00003  *
00004  *  Created on: Aug 10, 2025
00005  *      Author: elian
00006  */
00007
00008 #ifndef MODULES_GESTOR_ESTADOS_GESTORESTADOS_H_
00009 #define MODULES_GESTOR_ESTADOS_GESTORESTADOS_H_
00010
00025 typedef enum {
00026     STATE_INIT = 0,
00028
00029     STATE_IDLE,
00034
00035     STATE_RUNNING,
00038
00039     STATE_VEL_CHANGE,
00043
00044     STATE_BRAKING,
00047
00048     STATE_EMERGENCY
00052 } SystemState;
00053
00054
00064 typedef enum {
00065     ACTION_NONE = 0,
00066
00073     ACTION_INIT_DONE = 1,
00074
00081     ACTION_TO_IDLE,
00082
00091     ACTION_START,
00092
00101     ACTION_STOP,
00102
00108     ACTION_MOTOR_STOPPED,
00109
00116     ACTION_EMERGENCY,
00117
00127     ACTION_SET_FREC,
00128
00138     ACTION_SET_ACEL,
00139
00146     ACTION_SET_DESACEL,
00147
00157     ACTION_SET_DIR,
00158
00165     ACTION_TO_CONST_RUNNING,
00166
00173     ACTION_GET_FREC,
00174
00180     ACTION_GET_ACEL,
00181
00187     ACTION_GET_DESACEL,
00188
00194     ACTION_GET_DIR,
```

```
00195
00201     ACTION_IS_MOTOR_STOP,
00202 } SystemAction;
00203
00212 typedef enum {
00213     ACTION_RESP_OK = 0,
00215
00216     ACTION_RESP_ERR = 1,
00219
00220     ACTION_RESP_MOVING = 2,
00225
00226     ACTION_RESP_NOT_MOVING = 3,
00229
00230     ACTION_RESP_OUT_RANGE = 4,
00233
00234     ACTION_RESP_EMERGENCY_ACTIVE = 5
00236 } SystemActionResponse;
00237
00238
00273 SystemActionResponse GestorEstados_Action(SystemAction sysAct, int value);
00274
00275
00276 #endif /* MODULES_GESTOR_ESTADOS_GESTORESTADOS_H_ */
```

9.11. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_SVM/GestorSVM.c

```
#include <stdio.h>
#include "GestorSVM.h"
#include "stm32f103xb.h"
#include "../Inc/main.h"
#include "../Gestor_Estados/GestorEstados.h"
#include "../Gestor_Timers/GestorTimers.h"
```

Gráfico de dependencias incluidas en GestorSVM.c:

Estructuras de datos

- struct DatoCalculado
- struct Parametros

  Parámetros "sombra" para sincronización atómica con el lazo de cálculo.

defines

- #define MAX_TICKS 256

  ARR efectivo del timer de switching (resolución 256 pasos).
- #define TICKS_DESHABILITAR_CANAL 280

  Valor mayor a ARR para forzar que un CCR nunca dispare (deshabilitar canal por "posición imposible").
- #define MIN_TICKS_DIF 5

  Mínima separación (en ticks) entre eventos para considerar que no hay interferencia entre t1/t2/t3.
- #define BUFFER_CALCULO_SIZE 3

  Buffer circular (productor: timer de cálculo; consumidor: timer de switching).

Constantes de cálculo t1/t2 por regresión lineal

t1 y t2 se obtienen con una regresión lineal en función del ángulo parcial y el índice de modulación.

- #define CONST_CALC_T1_PROP (float)-3.59145E-6
- #define CONST_CALC_T1_ORD_ORG (float)224.2845426
- #define CONST_CALC_T2_PROP (float)+3.59145E-6
- #define CONST_CALC_T2_ORD_ORG (float)8.797345358

Enumeraciones

- enum CasoInterferenciaTimer { INTERF_NULA = 0 , INTERF_T1T2 , INTERF_T1T3 , INTERF_T2T3 , INTERF_T1T2_T2T3 }

  Casos de interferencia temporal entre eventos t1/t2/t3 (demasiado cercanos).

Funciones

- static void GestorSVM_Calculoaceleracioneracion (void)

  Aplica la rampa de velocidad (aceleracion/desaceleracion) sobre frecuenciaSalida y actualiza flags/estado.
- static int pinMap (int x)

  Convierte el XOR de estados (bit U/V/W) al índice interno {0,1,2}.
- static void GestorSVM_CalcularValoresSwitching (void)

  Calcula t1, t2 y t0 (y casos de interferencia) y los deja en ticksChannel[].
- static void GestorSVM_SwitchInterrupt (SwitchInterruptType intType)

  Handler interno llamado por el ISR de TIM3 según la fuente (CH1..CH4, RESET, CLEAN).
- static void GestorSVM_SwitchPuertos (OrdenSwitch orden, char estado, int numCuadrante)

  Escribe en BSRR los pines correspondientes a la orden y cuadrante.
- void GestorSVM_SetConfiguration (ConfiguracionSVM *configuracion)

  Carga la configuración base de SVM y prepara tablas de BSRR por cuadrante.
- void GestorSVM_CalcInterrupt ()

  Handler llamado por el timer de cálculo (productor). Encola t1/t2/t3 si hay espacio.
- int GestorSVM_SetFrec (int frec)

  Solicita nueva frecuencia objetivo (Hz). Inicia rampa si el motor está en marcha.
- int GestorSVM_SetDir (int dir)

  Cambia el sentido de giro si el motor está detenido.
- int GestorSVM_Setaceleracion (int nuevaaceleracion)

  Actualiza acelerada [Hz/s].
- int GestorSVM_SetDecel (int nuevaDecel)

  Actualiza desacelerada [Hz/s].
- int GestorSVM_MotorStart ()

  Arranca el motor: habilita drivers, inicia timers y rampa hacia frecuenciaReferenica.
- int GestorSVM_MotorStop ()

  Ordena frenado con rampa de desaceleracion hasta 0 Hz.
- int GestorSVM_Estop ()

  Parada de emergencia inmediata: desactiva timers, apaga drivers y salidas.
- int GestorSVM_GetFrec ()

  Devuelve la frecuencia de referencia configurada (Hz).
- int GestorSVM_Getaceleracion ()

  Devuelve acelerada actual [Hz/s].
- int GestorSVM_GetDesaceleracion ()

  Devuelve desacelerada actual [Hz/s].
- int GestorSVM_GetDir ()

  Devuelve sentido de giro (1 horario, -1 antihorario).
- void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef *htim)

  Callback HAL de Output Compare: enruta eventos CCR de TIM3 a GestorSVM_SwitchInterrupt.

Variables

- static int frecuenciaSwitching

  Frecuencia de switching [Hz] (TIM3).
- static int direccionRotacion = 1

  Sentido de rotación: 1 horario, -1 antihorario.
- static int32_t frecuenciaSalida

  Frecuencia de salida INSTANTÁNEA (escalada ×1e6) usada por el lazo de rampa.
- static uint32_t anguloSwitching

  Incremento angular por switching (grados×1e3).
- static volatile int indiceModulacion

  Índice de modulación (0..100). Controla la tensión de salida. La relación v/f debe ser constante.
- static int aceleracion

  acelerada configurada [Hz/s].
- static int desaceleracion

  Desacelerada configurada [Hz/s].
- static uint32_t anguloActual

  Ángulo absoluto (grados×1e3).
- static int32_t anguloParcial

  Ángulo parcial 0..60 ° (grados×1e3), usado para t1/t2. Puede ser negativo durante el diente.
- static volatile int cuadranteActual

  Cuadrante SVM actual (0..5).
- const int vectorSecuenciaPorCuadrante [6][7]

  Secuencia SVM por cuadrante (V0→V1→V2→V7→V2→V1→V0).
- int pinTogglePorCuadranteYOrden [6][3]

  Mapa de qué pin conmuta en cada orden y cuadrante.
- uint32_t estadoGPIOPorCuadranteYOrden [6][2]

  Palabras BSRR precalculadas por cuadrante/estado (0/1).
- uint32_t estadoGPIOOff

  BSRR para apagar U/V/W simultáneo.
- uint32_t estadoGPIOOn

  BSRR para encender U/V/W simultáneo.
- static volatile char flagAscensoAnguloParcial = 1

  Bandera de rampa del ángulo parcial (subida/bajada en diente).
- static volatile char estadoLogicoSalida [3]

  Estado lógico de salidas U/V/W (para depuración).
- static volatile int ticksChannel [3]

  t1/t2/t3 en ticks para el ciclo actual.
- static volatile CasoInterferenciaTimer casoInterferencia

  Caso de interferencia detectado para el ciclo actual.
- static volatile int32_t frecObjetivo

  Frecuencia objetivo (target) escalada ×1e6.
- static volatile int flagChangingFrecuencia

  Indica cambio de frecuencia en curso (rampa activa).
- static volatile int flagEsAcelerado

  1 si la rampa corresponde a una aceleración, 0 para desaceleración.
- static volatile int flagMotorRunning

  1 si el motor está en movimiento.
- static volatile int frecuenciaReferenica

  Frecuencia “de referencia” reportada (Hz).
- static volatile uint32_t cambioFrecuenciaPorCiclo

Delta por ciclo de switching (escalado ×1e6) para la rampa.

- volatile ValoresSwitching valoresSwitching

  Estructura auxiliar de switching usada por el ISR.
- volatile DatoCalculado bufferCalculo
- volatile Parametros paramSombra
- volatile int flagActualizarParamSombra

9.11.1. Documentación de «define»

9.11.1.1. BUFFER_CALCULO_SIZE

```
#define BUFFER_CALCULO_SIZE 3
```

Buffer circular (productor: timer de cálculo; consumidor: timer de switching).

Tamaño fijo 3: el productor se detiene si está lleno.

Definición en la línea 132 del archivo GestorSVM.c.

9.11.1.2. CONST_CALC_T1_ORD_ORG

```
#define CONST_CALC_T1_ORD_ORG (float)224.2845426
```

Definición en la línea 44 del archivo GestorSVM.c.

9.11.1.3. CONST_CALC_T1_PROP

```
#define CONST_CALC_T1_PROP (float)-3.59145E-6
```

Definición en la línea 43 del archivo GestorSVM.c.

9.11.1.4. CONST_CALC_T2_ORD_ORG

```
#define CONST_CALC_T2_ORD_ORG (float)8.797345358
```

Definición en la línea 46 del archivo GestorSVM.c.

9.11.1.5. CONST_CALC_T2_PROP

```
#define CONST_CALC_T2_PROP (float)+3.59145E-6
```

Definición en la línea 45 del archivo GestorSVM.c.

9.11.1.6. MAX_TICKS

```
#define MAX_TICKS 256
```

ARR efectivo del timer de switching (resolución 256 pasos).

Definición en la línea 26 del archivo GestorSVM.c.

9.11.1.7. MIN_TICKS_DIF

```
#define MIN_TICKS_DIF 5
```

Mínima separación (en ticks) entre eventos para considerar que no hay interferencia entre t1/t2/t3.

Definición en la línea 38 del archivo GestorSVM.c.

9.11.1.8. TICKS_DESHABILITAR_CANAL

```
#define TICKS_DESHABILITAR_CANAL 280
```

Valor mayor a ARR para forzar que un CCR nunca dispare (deshabilitar canal por "posición imposible").

Definición en la línea 32 del archivo GestorSVM.c.

9.11.2. Documentación de enumeraciones

9.11.2.1. CasoInterferenciaTimer

```
enum CasoInterferenciaTimer
```

Casos de interferencia temporal entre eventos t1/t2/t3 (demasiado cercanos).

Valores de enumeraciones

INTERF_NULA	
INTERF_T1T2	Sin interferencias.
INTERF_T1T3	t1 y t2 demasiado cercanos.
INTERF_T2T3	t1 y t3 demasiado cercanos.
INTERF_T1T2_T2T3	t2 y t3 demasiado cercanos. t1∼t2 y t2∼t3 simultáneamente.

Definición en la línea 53 del archivo GestorSVM.c.

9.11.3. Documentación de funciones

9.11.3.1. GestorSVM_CalcInterrupt()

```
void GestorSVM_CalcInterrupt (
            void )
```

Handler llamado por el timer de cálculo (productor). Encola t1/t2/t3 si hay espacio.

ISR (o handler llamado por ISR) del timer de cálculo.

Corre a 2× FREC_SWITCH (según tu diseño) para anticipar cómputos: si el buffer circular tiene espacio, calcula t1/t2/t3, cuadrante y posibles interferencias, y los encola para que el timer de switching (TIM3) los consuma. No bloquea si el buffer está lleno.

Nota

Tamaño de buffer: 3 entradas (pipeline productor/consumidor).

Definición en la línea 640 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.2. GestorSVM_CalcularValoresSwitching()

```
void GestorSVM_CalcularValoresSwitching (
            void )  [static]
```

Calcula t1, t2 y t0 (y casos de interferencia) y los deja en ticksChannel[].

- Actualiza ángulos (anguloActual y anguloParcial) y cuadranteActual.
- t1/t2 por regresión lineal (constantes CONST_CALC_T∗ y indiceModulacion).
- t0 = (255 - t1 - t2)/2. Luego: ticksC1=t0, ticksC2=t0+t1, ticksC3=t0+t1+t2.
- Clasifica interferencias según MIN_TICKS_DIF y setea casoInterferencia.

Definición en la línea 238 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.3. GestorSVM_Calculoaceleracioneracion()

```
void GestorSVM_Calculoaceleracioneracion (
            void )  [static]
```

Aplica la rampa de velocidad (aceleracion/desaceleracion) sobre frecuenciaSalida y actualiza flags/estado.

- Si flagActualizarParamSombra está activo, copia los parámetros de paramSombra.
- Ajusta frecuenciaSalida en ± cambioFrecuenciaPorCiclo hasta llegar a frecObjetivo.
- Al llegar a 0 Hz: detiene timers, limpia buffer, apaga GPIO y notifica ACTION_MOTOR_STOPPED.

  Nota

  En esta funcion se puede llegar a dar una incoherencia debido a la sincronizacion de las variables sombra. Si se esta ejecutando un cambio de frecuencia en la funcion GestorSVM_SetFrecOut, es decir que se habia escrito flagChangingFrec y antes de llegar levantar el flagActualizarParamSombra, justo en ese momento llega una interrupcion de este timer de calculo y en esa iteracion se terminaba de llegar a la velocidad taget. En ese caso se va a pisar ese flagChangingFrec y se va a quedar como si nunca hubiese pasado. Debido a que es un caso muy extremo no se analiza

Definición en la línea 513 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.4. GestorSVM_Estop()

```
int GestorSVM_Estop (
            void )
```

Parada de emergencia inmediata: desactiva timers, apaga drivers y salidas.

Parada de emergencia inmediata.

Devuelve

Siempre 0.

Siempre 0.

Deshabilita timers/interrupts, apaga drivers y salidas sin rampa.

Atención

Uso ante condiciones de falla. Requiere nueva orden de arranque.

Definición en la línea 857 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.5. GestorSVM_Getaceleracion()

```
int GestorSVM_Getaceleracion ()
```

Devuelve acelerada actual [Hz/s].

Definición en la línea 892 del archivo GestorSVM.c.

9.11.3.6. GestorSVM_GetDesaceleracion()

```
int GestorSVM_GetDesaceleracion ()
```

Devuelve desacelerada actual [Hz/s].

Definición en la línea 897 del archivo GestorSVM.c.

9.11.3.7. GestorSVM_GetDir()

```
int GestorSVM_GetDir ()
```

Devuelve sentido de giro (1 horario, -1 antihorario).

Obtiene el sentido de giro actual (1 horario, -1 antihorario).

Definición en la línea 902 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.8. GestorSVM_GetFrec()

```
int GestorSVM_GetFrec (
            void )
```

Devuelve la frecuencia de referencia configurada (Hz).

Lee la frecuencia objetivo de referencia (no escalada).

Devuelve

frecuenciaReferenica.

Frecuencia de referencia [Hz].

Nota

Si querés la frecuencia *instantánea* en rampa, podrías exponer otra API.

Definición en la línea 887 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.9. GestorSVM_MotorStart()

```
int GestorSVM_MotorStart (
            void )
```

Arranca el motor: habilita drivers, inicia timers y rampa hacia frecuenciaReferenica.

Inicia la marcha: arranca rampa de aceleración hacia frecReferencia.

Devuelve

0 si arranca; 1 si ya estaba en marcha.

0 si inicia correctamente, 1 si ya estaba en marcha.

Habilita drivers, inicia timers (cálculo y switching), precarga ticks, y pone flags para rampa ascendente. La frecuencia objetivo proviene de GestorSVM_SetFrec o GestorSVM_SetConfiguration.

Definición en la línea 794 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.10. GestorSVM_MotorStop()

```
int GestorSVM_MotorStop (
            void )
```

Ordena frenado con rampa de desaceleracion hasta 0 Hz.

Ordena frenado controlado (rampa de desacel hasta 0 Hz).

Devuelve

0 si acepta; 1 si ya estaba detenido.

0 si se acepta la orden, 1 si ya estaba detenido.

No corta drivers instantáneamente: mantiene el switching hasta llegar a 0 Hz, luego apaga salidas y notifica al gestor de estados.

Definición en la línea 831 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.11. GestorSVM_Setaceleracion()

```
int GestorSVM_Setaceleracion (
            int nuevaaceleracion)
```

Actualiza acelerada [Hz/s].

Devuelve

0 OK; -1 fuera de rango; -2 si hay rampa activa.

Definición en la línea 762 del archivo GestorSVM.c.

9.11.3.12. GestorSVM_SetConfiguration()

```
void GestorSVM_SetConfiguration (
            ConfiguracionSVM * configuracion)
```

Carga la configuración base de SVM y prepara tablas de BSRR por cuadrante.

Carga la configuración base del SVM.

Parámetros

configuracion	Puntero a ConfiguracionSVM.
configuracion	Puntero a ConfiguracionSVM con frec_switch, frecReferencia, direccionRotacion, acel y desacel.

Aplica frec_switch, rampas y sentido; y propaga frecReferencia al pipeline de cambio de velocidad. Internamente recalcula lookups/BSRR por cuadrante.

Atención

Validar rangos antes de invocar si la config proviene de UI/externo.

Definición en la línea 585 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.13. GestorSVM_SetDecel()

```
int GestorSVM_SetDecel (
            int decel)
```

Actualiza desacelerada [Hz/s].

Actualiza la desaceleración dinámica [Hz/seg].

Devuelve

0 OK; -1 fuera de rango; -2 si hay rampa activa.

Parámetros

decel	Nueva desaceleración.

Devuelve

0 OK; -1 fuera de rango; -2 si hay cambio de velocidad en curso.

Definición en la línea 778 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.14. GestorSVM_SetDir()

```
int GestorSVM_SetDir (
            int dir)
```

Cambia el sentido de giro si el motor está detenido.

Cambia el sentido de giro (solo con motor detenido).

Parámetros

dir	0 o 1 (mapeo a antihorario/horario).

Devuelve

0 OK; -1 fuera de rango; -2 si motor en marcha o rampa activa.

Parámetros

dir	0 o 1 (mapea a antihorario/horario según tu implementación).

Devuelve

- 0: Modificado.
- -1: Fuera de rango.
- -2: Rechazado (motor en marcha o cambio en curso).

Recalcula la tabla BSRR por cuadrante para invertir U/V.

Definición en la línea 717 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.15. GestorSVM_SetFrec()

```
int GestorSVM_SetFrec (
            int frec)
```

Solicita nueva frecuencia objetivo (Hz). Inicia rampa si el motor está en marcha.

Solicita una nueva frecuencia objetivo.

Parámetros

frec	Frecuencia objetivo [Hz].

Devuelve

0 si aceptada (motor detenido), 1 si aceptada con rampa en curso; -1 fuera de rango; -2 misma que la actual.

Parámetros

frec	Frecuencia objetivo [Hz].

Devuelve

- 0: Aceptada con motor detenido (queda como referencia).
- 1: Aceptada con motor en marcha (inicia rampa).
- -1: Fuera de rango (FERC_OUT_MIN..FERC_OUT_MAX).
- -2: Igual a la frecuencia actual (sin cambios).

Calcula delta por ciclo a partir de acel/ desacel y frec_switch, y actualiza parámetros sombra para el lazo de cálculo.

Definición en la línea 668 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.16. GestorSVM_SwitchInterrupt()

```
void GestorSVM_SwitchInterrupt (
            SwitchInterruptType intType)  [static]
```

Handler interno llamado por el ISR de TIM3 según la fuente (CH1..CH4, RESET, CLEAN).

Parámetros

intType	Fuente de interrupción SwitchInterruptType.

- RESET: consume una entrada del buffer y precarga CCR2/CCR3/CCR4.
- CH1/CH2/CH3: conmuta GPIO según orden/fase y maneja interferencias habilitando/deshabilitando CCxIE.
- CLEAN: limpia flags y estado interno del switching.

Definición en la línea 300 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.11.3.17. GestorSVM_SwitchPuertos()

```
void GestorSVM_SwitchPuertos (
            OrdenSwitch orden,
            char estado,
            int numCuadrante)  [static]
```

Escribe en BSRR los pines correspondientes a la orden y cuadrante.

Parámetros

orden	Orden de conmutación OrdenSwitch.
estado	0=primer estado del sector, 1=segundo estado (usa estadoGPIOPorCuadranteYOrden).
numCuadrante	Sector SVM (0..5).

Definición en la línea 496 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.3.18. HAL_TIM_OC_DelayElapsedCallback()

```
void HAL_TIM_OC_DelayElapsedCallback (
            TIM_HandleTypeDef * htim)
```

Callback HAL de Output Compare: enruta eventos CCR de TIM3 a GestorSVM_SwitchInterrupt.

Parámetros

htim	Puntero al handle del timer que disparó el evento.

- CH1 → SWITCH_INT_RESET
- CH2 → SWITCH_INT_CH1
- CH3 → SWITCH_INT_CH2
- CH4 → SWITCH_INT_CH3

Definición en la línea 918 del archivo GestorSVM.c.

Gráfico de llamadas de esta función:

9.11.3.19. pinMap()

```
int pinMap (
            int x)  [static]
```

Convierte el XOR de estados (bit U/V/W) al índice interno {0,1,2}.

Parámetros

x	Máscara de pin que cambia (1bit activo entre U/V/W).

Devuelve

Índice 0→U, 1→V, 2→W.

Definición en la línea 223 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.11.4. Documentación de variables

9.11.4.1. aceleracion

```
int aceleracion  [static]
```

acelerada configurada [Hz/s].

Definición en la línea 72 del archivo GestorSVM.c.

9.11.4.2. anguloActual

```
uint32_t anguloActual  [static]
```

Ángulo absoluto (grados×1e3).

Definición en la línea 76 del archivo GestorSVM.c.

9.11.4.3. anguloParcial

```
int32_t anguloParcial  [static]
```

Ángulo parcial 0..60° (grados×1e3), usado para t1/t2. Puede ser negativo durante el diente.

Definición en la línea 78 del archivo GestorSVM.c.

9.11.4.4. anguloSwitching

```
uint32_t anguloSwitching  [static]
```

Incremento angular por switching (grados×1e3).

Definición en la línea 68 del archivo GestorSVM.c.

9.11.4.5. bufferCalculo

```
volatile DatoCalculado bufferCalculo
```

Definición en la línea 145 del archivo GestorSVM.c.

9.11.4.6. cambioFrecuenciaPorCiclo

```
volatile uint32_t cambioFrecuenciaPorCiclo  [static]
```

Delta por ciclo de switching (escalado ×1e6) para la rampa.

Definición en la línea 124 del archivo GestorSVM.c.

9.11.4.7. casoInterferencia

```
volatile CasoInterferenciaTimer casoInterferencia  [static]
```

Caso de interferencia detectado para el ciclo actual.

Definición en la línea 111 del archivo GestorSVM.c.

9.11.4.8. cuadranteActual

```
volatile int cuadranteActual  [static]
```

Cuadrante SVM actual (0..5).

Definición en la línea 80 del archivo GestorSVM.c.

9.11.4.9. desaceleracion

```
int desaceleracion  [static]
```

Desacelerada configurada [Hz/s].

Definición en la línea 74 del archivo GestorSVM.c.

9.11.4.10. direccionRotacion

```
int direccionRotacion = 1  [static]
```

Sentido de rotación: 1 horario, -1 antihorario.

Definición en la línea 64 del archivo GestorSVM.c.

9.11.4.11. estadoGPIOOff

```
uint32_t estadoGPIOOff
```

BSRR para apagar U/V/W simultáneo.

Definición en la línea 100 del archivo GestorSVM.c.

9.11.4.12. estadoGPIOOn

```
uint32_t estadoGPIOOn
```

BSRR para encender U/V/W simultáneo.

Definición en la línea 102 del archivo GestorSVM.c.

9.11.4.13. estadoGPIOPorCuadranteYOrden

```
uint32_t estadoGPIOPorCuadranteYOrden[6][2]
```

Palabras BSRR precalculadas por cuadrante/estado (0/1).

Definición en la línea 98 del archivo GestorSVM.c.

9.11.4.14. estadoLogicoSalida

```
volatile char estadoLogicoSalida[3]  [static]
```

Estado lógico de salidas U/V/W (para depuración).

Definición en la línea 107 del archivo GestorSVM.c.

9.11.4.15. flagActualizarParamSombra

```
volatile int flagActualizarParamSombra
```

Definición en la línea 160 del archivo GestorSVM.c.

9.11.4.16. flagAscensoAnguloParcial

```
volatile char flagAscensoAnguloParcial = 1  [static]
```

Bandera de rampa del ángulo parcial (subida/bajada en diente).

Definición en la línea 105 del archivo GestorSVM.c.

9.11.4.17. flagChangingFrecuencia

```
volatile int flagChangingFrecuencia  [static]
```

Indica cambio de frecuencia en curso (rampa activa).

Definición en la línea 116 del archivo GestorSVM.c.

9.11.4.18. flagEsAcelerado

```
volatile int flagEsAcelerado  [static]
```

1 si la rampa corresponde a una aceleración, 0 para desaceleración.

Definición en la línea 118 del archivo GestorSVM.c.

9.11.4.19. flagMotorRunning

```
volatile int flagMotorRunning  [static]
```

1 si el motor está en movimiento.

Definición en la línea 120 del archivo GestorSVM.c.

9.11.4.20. frecObjetivo

```
volatile int32_t frecObjetivo  [static]
```

Frecuencia objetivo (target) escalada ×1e6.

Definición en la línea 114 del archivo GestorSVM.c.

9.11.4.21. frecuenciaReferenica

```
volatile int frecuenciaReferenica  [static]
```

Frecuencia “de referencia” reportada (Hz).

Definición en la línea 122 del archivo GestorSVM.c.

9.11.4.22. frecuenciaSalida

```
int32_t frecuenciaSalida  [static]
```

Frecuencia de salida INSTANTÁNEA (escalada ×1e6) usada por el lazo de rampa.

Definición en la línea 66 del archivo GestorSVM.c.

9.11.4.23. frecuenciaSwitching

```
int frecuenciaSwitching  [static]
```

Frecuencia de switching [Hz] (TIM3).

Definición en la línea 62 del archivo GestorSVM.c.

9.11.4.24. indiceModulacion

```
volatile int indiceModulacion  [static]
```

Índice de modulación (0..100). Controla la tensión de salida. La relación v/f debe ser constante.

Definición en la línea 70 del archivo GestorSVM.c.

9.11.4.25. paramSombra

```
volatile Parametros paramSombra
```

Definición en la línea 159 del archivo GestorSVM.c.

9.11.4.26. pinTogglePorCuadranteYOrden

```
int pinTogglePorCuadranteYOrden[6][3]
```

Mapa de qué pin conmuta en cada orden y cuadrante.

Definición en la línea 96 del archivo GestorSVM.c.

9.11.4.27. ticksChannel

```
volatile int ticksChannel[3]  [static]
```

t1/t2/t3 en ticks para el ciclo actual.

Definición en la línea 109 del archivo GestorSVM.c.

9.11.4.28. valoresSwitching

```
volatile ValoresSwitching valoresSwitching
```

Estructura auxiliar de switching usada por el ISR.

Definición en la línea 126 del archivo GestorSVM.c.

9.11.4.29. vectorSecuenciaPorCuadrante

```
const int vectorSecuenciaPorCuadrante[6][7]
```

Valor inicial:

```
= {
    {0b000,0b100,0b110,0b111,0b110,0b100,0b000},
    {0b000,0b010,0b110,0b111,0b110,0b010,0b000},
    {0b000,0b010,0b011,0b111,0b011,0b010,0b000},
    {0b000,0b001,0b011,0b111,0b011,0b001,0b000},
    {0b000,0b001,0b101,0b111,0b101,0b001,0b000},
    {0b000,0b100,0b101,0b111,0b101,0b100,0b000}
}
```

Secuencia SVM por cuadrante (V0→V1→V2→V7→V2→V1→V0).

Cada fila representa el vector lógico {U,V,W} en 7 pasos por sector.

Definición en la línea 86 del archivo GestorSVM.c.

9.12. GestorSVM.c

Ir a la documentación de este archivo.

```
00001
00014
00015 #include <stdio.h>
00016 #include "GestorSVM.h"
00017 #include "stm32f103xb.h"
00018 #include "../Inc/main.h"
00019 #include "../Gestor_Estados/GestorEstados.h"
00020 #include "../Gestor_Timers/GestorTimers.h"
00021
00026 #define MAX_TICKS 256
00027
00032 #define TICKS_DESHABILITAR_CANAL 280
00033
00038 #define MIN_TICKS_DIF 5
00039
00043 #define CONST_CALC_T1_PROP      (float)-3.59145E-6
00044 #define CONST_CALC_T1_ORD_ORG   (float)224.2845426
00045 #define CONST_CALC_T2_PROP      (float)+3.59145E-6
00046 #define CONST_CALC_T2_ORD_ORG   (float)8.797345358
00048
00053 typedef enum {
00054     INTERF_NULA = 0,
00055     INTERF_T1T2,
00056     INTERF_T1T3,
00057     INTERF_T2T3,
00058     INTERF_T1T2_T2T3
00059 } CasoInterferenciaTimer;
00060
00062 static int frecuenciaSwitching;
00064 static int direccionRotacion = 1;
00066 static int32_t frecuenciaSalida;
00068 static uint32_t anguloSwitching;
00070 static volatile int indiceModulacion;
00072 static int aceleracion;
00074 static int desaceleracion;
00076 static uint32_t anguloActual;
00078 static int32_t anguloParcial;
00080 static volatile int cuadranteActual;
00081
00086 const int vectorSecuenciaPorCuadrante[6][7] = {
00087     {0b000,0b100,0b110,0b111,0b110,0b100,0b000}, // Sector 1
00088     {0b000,0b010,0b110,0b111,0b110,0b010,0b000}, // Sector 2
00089     {0b000,0b010,0b011,0b111,0b011,0b010,0b000}, // Sector 3
00090     {0b000,0b001,0b011,0b111,0b011,0b001,0b000}, // Sector 4
00091     {0b000,0b001,0b101,0b111,0b101,0b001,0b000}, // Sector 5
00092     {0b000,0b100,0b101,0b111,0b101,0b100,0b000}  // Sector 6
00093 };
00094
00096 int pinTogglePorCuadranteYOrden[6][3];
00098 uint32_t estadoGPIOPorCuadranteYOrden[6][2];
00100 uint32_t estadoGPIOOff;
00102 uint32_t estadoGPIOOn;
```

```
00103
00105 static volatile char flagAscensoAnguloParcial = 1;
00107 static volatile char estadoLogicoSalida[3];
00109 static volatile int ticksChannel[3];
00111 static volatile CasoInterferenciaTimer casoInterferencia;
00112
00114 static volatile int32_t frecObjetivo;
00116 static volatile int flagChangingFrecuencia;
00118 static volatile int flagEsAcelerado;
00120 static volatile int flagMotorRunning;
00122 static volatile int frecuenciaReferenica;
00124 static volatile uint32_t cambioFrecuenciaPorCiclo;
00126 volatile ValoresSwitching valoresSwitching;
00127
00132 #define BUFFER_CALCULO_SIZE 3
00133
00134 typedef struct {
00135     int ticksChannel1[BUFFER_CALCULO_SIZE];
00136     int ticksChannel2[BUFFER_CALCULO_SIZE];
00137     int ticksChannel3[BUFFER_CALCULO_SIZE];
00138     CasoInterferenciaTimer casoInterferencia[BUFFER_CALCULO_SIZE];
00139     int cuadranteActual[BUFFER_CALCULO_SIZE];
00140     int indiceEscritura;
00141     int indiceLectura;
00142     int contadorDeDatos;
00143 } DatoCalculado;
00144
00145 volatile DatoCalculado bufferCalculo;
00146
00150 typedef struct {
00151     int32_t frecObjetivo;
00152     uint32_t cambioFrecuenciaPorCiclo;
00153     int direccionRotacion;
00154     int flagMotorRunning;
00155     int flagEsAcelerado;
00156     int flagChangingFrecuencia;
00157 } Parametros;
00158
00159 volatile Parametros paramSombra;
00160 volatile int flagActualizarParamSombra;
00161
00162 /* ================================ Prototipos privados ================================ */
00163
00180 static void GestorSVM_Calculoaceleracioneracion(void);
00181
00188 static int pinMap(int x);
00189
00199 static void GestorSVM_CalcularValoresSwitching(void);
00200
00210 static void GestorSVM_SwitchInterrupt(SwitchInterruptType intType);
00211
00219 static void GestorSVM_SwitchPuertos(OrdenSwitch orden, char estado, int numCuadrante);
00220
00221 /* ================================ Implementación privada ================================ */
00222
00223 static int pinMap(int x) {
00224     int r = 0;
00225     x &= 0x07; // limitar a 3 bits (U,V,W)
00226     if (x == 1) {
00227         r = 2;   // W
00228     }
00229     else if (x == 2) {
00230         r = 1;   // V
00231     }
00232     else if (x == 4) {
00233         r = 0;   // U
00234     }
00235     return r;
00236 }
00237
00238 static void GestorSVM_CalcularValoresSwitching() {
00239     int t1, t2, t0;
00240     int ticksC1, ticksC2, ticksC3;
00241
00242     /* Rampa de velocidad si corresponde */
00243     if (flagChangingFrecuencia) {
00244         GestorSVM_Calculoaceleracioneracion();
00245     }
00246
00247     /* Avance angular y gestión de diente 0..60° */
00248     anguloActual += anguloSwitching;
00249     if (anguloActual >= 360 * 1000 * 1000) {
00250         anguloActual -= 360 * 1000 * 1000;
00251         cuadranteActual = 0;
00252         flagAscensoAnguloParcial = 1;
00253         anguloParcial = anguloActual;
00254     } else {
```

```
00255         if (flagAscensoAnguloParcial == 1) {
00256             anguloParcial += anguloSwitching;
00257             if (anguloParcial >= 60 * 1000 * 1000) {
00258                 anguloParcial = 2 * 60 * 1000 * 1000 - anguloParcial; // espejo
00259                 flagAscensoAnguloParcial = 0;
00260                 cuadranteActual = (cuadranteActual + 1) % 6;
00261             }
00262         } else {
00263             anguloParcial -= anguloSwitching;
00264             if (anguloParcial <= 0) {
00265                 anguloParcial = -anguloParcial; // espejo
00266                 flagAscensoAnguloParcial = 1;
00267                 cuadranteActual = (cuadranteActual + 1) % 6;
00268             }
00269         }
00270     }
00271
00272     /* t1/t2 por regresión e índice de modulación */
00273     t1 = (int)(((CONST_CALC_T1_PROP * (float)anguloParcial + CONST_CALC_T1_ORD_ORG) *
      indiceModulacion) / 100);
00274     t2 = (int)(((CONST_CALC_T2_PROP * (float)anguloParcial + CONST_CALC_T2_ORD_ORG) *
      indiceModulacion) / 100);
00275     t0 = (int)((float)(255 - t1 - t2) * 0.5);
00276
00277     /* Ticks absolutos en el período */
00278     ticksC1 = t0;
00279     ticksC2 = t0 + t1;
00280     ticksC3 = t0 + t1 + t2;
00281
00282     /* Clasificación de interferencias */
00283     if (ticksC3 - ticksC1 < MIN_TICKS_DIF) {
00284         casoInterferencia = INTERF_T1T3;
00285     } else if (ticksC2 - ticksC1 < MIN_TICKS_DIF && ticksC3 - ticksC2 < MIN_TICKS_DIF) {
00286         casoInterferencia = INTERF_T1T2_T2T3;
00287     } else if (ticksC2 - ticksC1 < MIN_TICKS_DIF) {
00288         casoInterferencia = INTERF_T1T2;
00289     } else if (ticksC3 - ticksC2 < MIN_TICKS_DIF) {
00290         casoInterferencia = INTERF_T2T3;
00291     } else {
00292         casoInterferencia = INTERF_NULA;
00293     }
00294
00295     ticksChannel[0] = ticksC1;
00296     ticksChannel[1] = ticksC2;
00297     ticksChannel[2] = ticksC3;
00298 }
00299
00300 static void GestorSVM_SwitchInterrupt(SwitchInterruptType intType) {
00301     volatile static int countUpTx[3];              // 1=subiendo, 0=bajando
00302     volatile static int flagTxActivo[3];           // espera de evento por canal
00303     volatile static CasoInterferenciaTimer interferencia;
00304     volatile static int cuadrante = 0;
00305     int ticks1, ticks2, ticks3;
00306
00307     /* Si no hay datos precargados, no hacer nada */
00308     if (bufferCalculo.contadorDeDatos <= 0) {
00309         return;
00310     }
00311
00312     switch (intType) {
00313         case SWITCH_INT_CH1:
00314             if (flagTxActivo[0]) {
00315                 flagTxActivo[0] = 0;
00316                 if (countUpTx[0]) {
00317                     GestorSVM_SwitchPuertos(ORDEN_SWITCH_1_UP, 1, cuadrante);
00318                     countUpTx[0] = 0;
00319                 } else {
00320                     countUpTx[0] = 1;
00321                     GestorSVM_SwitchPuertos(ORDEN_SWITCH_1_DOWN, 0, cuadrante);
00322                     /* Rehabilita interrupciones de los otros canales */
00323                     TIM3->DIER |= TIM_DIER_CC2IE;
00324                     TIM3->DIER |= TIM_DIER_CC3IE;
00325                     TIM3->DIER |= TIM_DIER_CC4IE;
00326                 }
00327             } else {
00328                 flagTxActivo[0] = 1;
00329             }
00330             break;
00331
00332         case SWITCH_INT_CH2:
00333             if (flagTxActivo[1]) {
00334                 flagTxActivo[1] = 0;
00335                 if (countUpTx[1]) {
00336                     GestorSVM_SwitchPuertos(ORDEN_SWITCH_2_UP, 1, cuadrante);
00337                     countUpTx[1] = 0;
00338                 } else {
00339                     GestorSVM_SwitchPuertos(ORDEN_SWITCH_2_DOWN, 0, cuadrante);
```

```
00340                   countUpTx[1] = 1;
00341               }
00342           } else {
00343               if (interferencia == INTERF_T1T2_T2T3) {
00344                   flagTxActivo[1] = 0;
00345                   TIM3->DIER &= ~TIM_DIER_CC3IE; // desactiva T2
00346               } else {
00347                   flagTxActivo[1] = 1;
00348               }
00349           }
00350           break;
00351
00352       case SWITCH_INT_CH3:
00353           if (flagTxActivo[2]) {
00354               flagTxActivo[2] = 0;
00355               if (countUpTx[2]) {
00356                   GestorSVM_SwitchPuertos(ORDEN_SWITCH_3_UP, 1, cuadrante);
00357
00358                   /* Manejo de interferencias */
00359                   switch (interferencia) {
00360                       case INTERF_NULA:
00361                           flagTxActivo[0] = 1;
00362                           flagTxActivo[1] = 1;
00363                           flagTxActivo[2] = 1;
00364                           break;
00365                       case INTERF_T1T2:
00366                           flagTxActivo[0] = 0;  // T1 pendiente
00367                           flagTxActivo[1] = 0;  // desactiva T2
00368                           flagTxActivo[2] = 1;
00369                           // Deshabilitacion T2
00370                           TIM3->DIER &= ~TIM_DIER_CC3IE;
00371                           // Activacion T1
00372                           TIM3->DIER |= TIM_DIER_CC2IE;
00373                           break;
00374                       case INTERF_T2T3:
00375                           flagTxActivo[0] = 1;
00376                           flagTxActivo[1] = 0;  // T2 pendiente
00377                           flagTxActivo[2] = 0;  // T3 off
00378                           TIM3->DIER &= ~TIM_DIER_CC4IE; // CC4 off
00379                           TIM3->DIER |=  TIM_DIER_CC3IE; // CC3 on
00380                           break;
00381                       case INTERF_T1T3:
00382                           /* No debe ocurrir aquí */
00383                           break;
00384                       case INTERF_T1T2_T2T3:
00385                           flagTxActivo[0] = 0;
00386                           flagTxActivo[1] = 0;
00387                           flagTxActivo[2] = 1;
00388                           TIM3->DIER &= ~TIM_DIER_CC3IE; // CC3 off
00389                           TIM3->DIER |=  TIM_DIER_CC2IE; // CC2 on
00390                           break;
00391                   }
00392                   /* Próximas interrupciones: cuenta abajo */
00393                   countUpTx[0] = 0;
00394                   countUpTx[1] = 0;
00395                   countUpTx[2] = 0;
00396               } else {
00397                   GestorSVM_SwitchPuertos(ORDEN_SWITCH_3_DOWN, 0, cuadrante);
00398                   countUpTx[2] = 1;
00399               }
00400           } else {
00401               if (interferencia == INTERF_T1T3) {
00402                   flagTxActivo[0] = 0;
00403                   flagTxActivo[1] = 0;
00404                   flagTxActivo[2] = 0;
00405               } else {
00406                   flagTxActivo[2] = 1;
00407               }
00408           }
00409           break;
00410
00411       case SWITCH_INT_RESET:
00412           /* Re-sync / precarga */
00413           countUpTx[0] = 1;
00414           countUpTx[1] = 1;
00415           countUpTx[2] = 1;
00416
00417           /* Consume entrada del buffer */
00418           ticks1 = bufferCalculo.ticksChannel1[bufferCalculo.indiceLectura];
00419           ticks2 = bufferCalculo.ticksChannel2[bufferCalculo.indiceLectura];
00420           ticks3 = bufferCalculo.ticksChannel3[bufferCalculo.indiceLectura];
00421           interferencia = bufferCalculo.casoInterferencia[bufferCalculo.indiceLectura];
00422           cuadrante = bufferCalculo.cuadranteActual[bufferCalculo.indiceLectura];
00423           bufferCalculo.indiceLectura = (bufferCalculo.indiceLectura + 1) % BUFFER_CALCULO_SIZE;
00424           bufferCalculo.contadorDeDatos--;
00425
00426           /* Habilitación según interferencias */
```

```
00427             switch (interferencia) {
00428                 case INTERF_NULA:
00429                     flagTxActivo[0] = 1;
00430                     flagTxActivo[1] = 1;
00431                     flagTxActivo[2] = 1;
00432                     TIM3->DIER |= TIM_DIER_CC2IE;
00433                     TIM3->DIER |= TIM_DIER_CC3IE;
00434                     TIM3->DIER |= TIM_DIER_CC4IE;
00435                     break;
00436                 case INTERF_T1T2:
00437                     flagTxActivo[0] = 0;
00438                     flagTxActivo[1] = 1;
00439                     flagTxActivo[2] = 1;
00440                     TIM3->DIER &= ~TIM_DIER_CC2IE;
00441                     TIM3->DIER |= TIM_DIER_CC3IE;
00442                     TIM3->DIER |= TIM_DIER_CC4IE;
00443                     break;
00444                 case INTERF_T2T3:
00445                     flagTxActivo[0] = 1;
00446                     flagTxActivo[1] = 0;
00447                     flagTxActivo[2] = 1;
00448                     TIM3->DIER |= TIM_DIER_CC2IE;
00449                     TIM3->DIER &= ~TIM_DIER_CC3IE;
00450                     TIM3->DIER |= TIM_DIER_CC4IE;
00451                     break;
00452                 case INTERF_T1T3:
00453                     flagTxActivo[0] = 0;
00454                     flagTxActivo[1] = 0;
00455                     flagTxActivo[2] = 0;
00456                     /* Carga posiciones distantes (nunca disparan simultáneo) */
00457                     ticks1 = 20;
00458                     ticks2 = 60;
00459                     ticks3 = 100;
00460                     TIM3->DIER |= TIM_DIER_CC2IE;
00461                     TIM3->DIER |= TIM_DIER_CC3IE;
00462                     TIM3->DIER |= TIM_DIER_CC4IE;
00463                     break;
00464                 case INTERF_T1T2_T2T3:
00465                     // Desactivacion de T1 y Activacion de T2 y T3
00466                     flagTxActivo[0] = 0;
00467                     flagTxActivo[1] = 1;
00468                     flagTxActivo[2] = 1;
00469                     TIM3->DIER &= ~TIM_DIER_CC2IE;
00470                     TIM3->DIER |= TIM_DIER_CC3IE;
00471                     TIM3->DIER |= TIM_DIER_CC4IE;
00472                     break;
00473             }
00474 
00475             /* Precarga CCRs (t1→CCR2, t2→CCR3, t3→CCR4) */
00476             TIM3->CCR2 = ticks1;
00477             TIM3->CCR3 = ticks2;
00478             TIM3->CCR4 = ticks3;
00479             break;
00480 
00481         case SWITCH_INT_CLEAN:
00482             /* Limpieza de estado */
00483             countUpTx[0] = 1;
00484             countUpTx[1] = 1;
00485             countUpTx[2] = 1;
00486             flagTxActivo[0] = 0;
00487             flagTxActivo[1] = 0;
00488             flagTxActivo[2] = 0;
00489             break;
00490 
00491         default:
00492             break;
00493     }
00494 }
00495 
00496 static void GestorSVM_SwitchPuertos(OrdenSwitch orden, char estado, int numCuadrante) {
00497     switch (orden) {
00498         case ORDEN_SWITCH_1_UP:
00499         case ORDEN_SWITCH_2_UP:
00500         case ORDEN_SWITCH_3_DOWN:
00501         case ORDEN_SWITCH_2_DOWN:
00502             GPIOA->BSRR = estadoGPIOPorCuadranteYOrden[numCuadrante][estado];
00503             break;
00504         case ORDEN_SWITCH_3_UP:
00505             GPIOA->BSRR = estadoGPIOOn;
00506             break;
00507         case ORDEN_SWITCH_1_DOWN:
00508             GPIOA->BSRR = estadoGPIOOff;
00509             break;
00510     }
00511 }
00512 
00513 static void GestorSVM_Calculoaceleracioneracion() {
```

```
00514     /* Sincronización con parámetros sombra, si corresponde */
00515     if (flagActualizarParamSombra) {
00516         frecObjetivo       = paramSombra.frecObjetivo;
00517         cambioFrecuenciaPorCiclo = paramSombra.cambioFrecuenciaPorCiclo;
00518         direccionRotacion  = paramSombra.direccionRotacion;
00519         flagMotorRunning   = paramSombra.flagMotorRunning;
00520         flagEsAcelerado    = paramSombra.flagEsAcelerado;
00521         flagChangingFrecuencia   = paramSombra.flagChangingFrecuencia;
00522         flagActualizarParamSombra = 0;
00523     }
00524 
00525     /* Aplicación de rampa */
00526     if (flagEsAcelerado) {
00527         frecuenciaSalida += cambioFrecuenciaPorCiclo;
00528         if (frecuenciaSalida >= frecObjetivo) {
00529             frecuenciaSalida = frecObjetivo;
00530             GestorEstados_Action(ACTION_TO_CONST_RUNNING, 0);
00531             flagChangingFrecuencia = 0;
00532         }
00533     } else {
00534         frecuenciaSalida -= cambioFrecuenciaPorCiclo;
00535         if (frecuenciaSalida <= frecObjetivo) {
00536             frecuenciaSalida = frecObjetivo;
00537 
00538             if (frecObjetivo == 0) {
00539                 flagMotorRunning = 0;
00540                 GestorTimers_DetenerTimerSVM();
00541                 GestorSVM_SwitchInterrupt(SWITCH_INT_CLEAN);
00542 
00543                 /* Limpia buffer productor/consumidor */
00544                 bufferCalculo.indiceEscritura = 0;
00545                 bufferCalculo.indiceLectura   = 0;
00546                 bufferCalculo.contadorDeDatos     = 0;
00547 
00548                 /* Apaga salidas y drivers */
00549                 HAL_GPIO_WritePin(GPIOA, GPIO_U_IN, GPIO_PIN_RESET);
00550                 HAL_GPIO_WritePin(GPIOA, GPIO_V_IN, GPIO_PIN_RESET);
00551                 HAL_GPIO_WritePin(GPIOA, GPIO_W_IN, GPIO_PIN_RESET);
00552 
00553                 HAL_GPIO_WritePin(GPIOA, GPIO_U_SD, GPIO_PIN_RESET);
00554                 HAL_GPIO_WritePin(GPIOA, GPIO_V_SD, GPIO_PIN_RESET);
00555                 HAL_GPIO_WritePin(GPIOA, GPIO_W_SD, GPIO_PIN_RESET);
00556 
00557                 /* Notifica detención */
00558                 GestorEstados_Action(ACTION_MOTOR_STOPPED, 0);
00559             } else {
00560                 GestorEstados_Action(ACTION_TO_CONST_RUNNING, 0);
00561             }
00562             flagChangingFrecuencia = 0;
00563         }
00564     }
00565 
00566     /* Índice de modulación (V/f) */
00567     if (frecuenciaSalida < 50 * 1000 * 1000) {
00568         indiceModulacion = frecuenciaSalida / (500 * 1000); // lineal hasta 50 Hz
00569     } else {
00570         indiceModulacion = 100;
00571     }
00572     if (indiceModulacion < 1) {
00573         indiceModulacion = 1;
00574     }
00575 
00576     /* Incremento angular por switching (cuidar orden para evitar overflow) */
00577     anguloSwitching = (frecuenciaSalida / frecuenciaSwitching) * 360;
00578 }
00579 
00585 void GestorSVM_SetConfiguration(ConfiguracionSVM* configuracion) {
00586     int i;
00587     /* Dinámicos */
00588     frecuenciaSwitching       = configuracion->frecuenciaSwitching;
00589     direccionRotacion = configuracion->direccionRotacion;
00590     aceleracion               = configuracion->aceleracion;
00591     desaceleracion            = configuracion->desaceleracion;
00592 
00593     /* Referencia (inicia como target) */
00594     GestorSVM_SetFrec(configuracion->frecuenciaReferenica);
00595 
00596     printf("Configuracion Seteada \n");
00597     int estado0, estado1, estado2, estado3;
00598     int pinToggle1, pinToggle2, pinToggle3;
00599     uint32_t myBSRR;
00600 
00601     estadoGPIOOff = (GPIO_U_IN | GPIO_V_IN | GPIO_W_IN) << 16;
00602     estadoGPIOOn  = (GPIO_U_IN | GPIO_V_IN | GPIO_W_IN);
00603 
00604     for (i = 0; i < 6; i++) {
00605         estado0 = vectorSecuenciaPorCuadrante[i][0];
```

```
00606         estado1 = vectorSecuenciaPorCuadrante[i][1];
00607         estado2 = vectorSecuenciaPorCuadrante[i][2];
00608         estado3 = vectorSecuenciaPorCuadrante[i][3];
00609 
00610         /* Estado 0→1 */
00611         myBSRR = 0;
00612         myBSRR |= ((estado1 & 0b100) > 0) ? puerto_senal_pierna[0] : (puerto_senal_pierna[0] << 16);
00613         myBSRR |= ((estado1 & 0b010) > 0) ? puerto_senal_pierna[1] : (puerto_senal_pierna[1] << 16);
00614         myBSRR |= ((estado1 & 0b001) > 0) ? puerto_senal_pierna[2] : (puerto_senal_pierna[2] << 16);
00615         estadoGPIOPorCuadranteYOrden[i][0] = myBSRR;
00616 
00617         /* Estado 1→2 */
00618         myBSRR = 0;
00619         myBSRR |= ((estado2 & 0b100) > 0) ? puerto_senal_pierna[0] : (puerto_senal_pierna[0] << 16);
00620         myBSRR |= ((estado2 & 0b010) > 0) ? puerto_senal_pierna[1] : (puerto_senal_pierna[1] << 16);
00621         myBSRR |= ((estado2 & 0b001) > 0) ? puerto_senal_pierna[2] : (puerto_senal_pierna[2] << 16);
00622         estadoGPIOPorCuadranteYOrden[i][1] = myBSRR;
00623 
00624         /* Pines que conmutan entre estados (XOR) */
00625         pinToggle1 = pinMap(estado0 ^ estado1);
00626         pinToggle2 = pinMap(estado1 ^ estado2);
00627         pinToggle3 = pinMap(estado2 ^ estado3);
00628 
00629         /* Mapa a U/V/W con desplazamiento */
00630         pinTogglePorCuadranteYOrden[i][0] = GPIO_U_IN << (pinToggle1 * 2);
00631         pinTogglePorCuadranteYOrden[i][1] = GPIO_U_IN << (pinToggle2 * 2);
00632         pinTogglePorCuadranteYOrden[i][2] = GPIO_U_IN << (pinToggle3 * 2);
00633     }
00634 }
00635 
00640 void GestorSVM_CalcInterrupt() {
00641     int indiceEscritura;
00642 
00643     if (bufferCalculo.contadorDeDatos < BUFFER_CALCULO_SIZE) {
00644         indiceEscritura = bufferCalculo.indiceEscritura;
00645 
00646         GestorSVM_CalcularValoresSwitching();
00647 
00648         /* Encolar resultados */
00649         bufferCalculo.ticksChannel1[indiceEscritura] = ticksChannel[0];
00650         bufferCalculo.ticksChannel2[indiceEscritura] = ticksChannel[1];
00651         bufferCalculo.ticksChannel3[indiceEscritura] = ticksChannel[2];
00652         bufferCalculo.casoInterferencia[indiceEscritura] = casoInterferencia;
00653         bufferCalculo.cuadranteActual[indiceEscritura]   = cuadranteActual;
00654 
00655         bufferCalculo.indiceEscritura = (indiceEscritura + 1) % BUFFER_CALCULO_SIZE;
00656         bufferCalculo.contadorDeDatos++;
00657     }
00658 }
00659 
00660 /* ------------------------ API invocada por el Gestor de Estados (SPI) ------------------------ */
00661 
00668 int GestorSVM_SetFrec(int frec) {
00669     int flagEsAcelerado_local;
00670     int32_t cambioFrecuenciaPorCiclo_local;
00671     int32_t frecTarget_local;
00672     int32_t nuevaFrec;
00673 
00674     if (frec < FERC_OUT_MIN || frec > FERC_OUT_MAX) {
00675         return -1;
00676     }
00677     nuevaFrec = (int32_t)frec * 1000 * 1000;
00678     if (frecuenciaSalida == nuevaFrec) {
00679         return -2;
00680     }
00681 
00682     if (flagMotorRunning) {
00683         /* Determinar sentido de la rampa */
00684         if (frecuenciaSalida < nuevaFrec) {
00685             flagEsAcelerado_local = 1;
00686             cambioFrecuenciaPorCiclo_local = (aceleracion * 1000 * 1000) / (frecuenciaSwitching);
00687         } else {
00688             flagEsAcelerado_local = 0;
00689             cambioFrecuenciaPorCiclo_local = (desaceleracion * 1000 * 1000) / (frecuenciaSwitching);
00690         }
00691 
00692         frecTarget_local = nuevaFrec;
00693         frecuenciaReferenica = frec;
00694 
00695         /* Parámetros sombra */
00696         paramSombra.flagMotorRunning   = 1;
00697         paramSombra.flagChangingFrecuencia   = 1;
00698         paramSombra.flagEsAcelerado    = flagEsAcelerado_local;
00699         paramSombra.cambioFrecuenciaPorCiclo = cambioFrecuenciaPorCiclo_local;
00700         paramSombra.frecObjetivo       = frecTarget_local;
00701         flagActualizarParamSombra = 1;
00702 
```

```
00703         flagChangingFrecuencia = 1;
00704         return 1;
00705     } else {
00706         frecuenciaReferenica = frec;
00707         return 0;
00708     }
00709 }
00710
00717 int GestorSVM_SetDir(int dir) {
00718     int estado1, estado2;
00719     int i;
00720     uint32_t myBSRR;
00721
00722     if (flagMotorRunning) {
00723         return -2;
00724     }
00725     if (flagChangingFrecuencia) {
00726         return -2;
00727     }
00728     if (dir != 0 && dir != 1) {
00729         return -1;
00730     }
00731     if (dir == direccionRotacion) {
00732         return 0;
00733     }
00734
00735     /* Recalcular tablas BSRR intercambiando UV */
00736     for (i = 0; i < 6; i++) {
00737         estado1 = vectorSecuenciaPorCuadrante[i][1];
00738         estado2 = vectorSecuenciaPorCuadrante[i][2];
00739
00740         myBSRR = 0;
00741         myBSRR |= ((estado1 & 0b100) ? GPIO_V_IN : (GPIO_V_IN << 16));
00742         myBSRR |= ((estado1 & 0b010) ? GPIO_U_IN : (GPIO_U_IN << 16));
00743         myBSRR |= ((estado1 & 0b001) ? GPIO_W_IN : (GPIO_W_IN << 16));
00744         estadoGPIOPorCuadranteYOrden[i][0] = myBSRR;
00745
00746         myBSRR = 0;
00747         myBSRR |= ((estado2 & 0b100) ? GPIO_V_IN : (GPIO_V_IN << 16));
00748         myBSRR |= ((estado2 & 0b010) ? GPIO_U_IN : (GPIO_U_IN << 16));
00749         myBSRR |= ((estado2 & 0b001) ? GPIO_W_IN : (GPIO_W_IN << 16));
00750         estadoGPIOPorCuadranteYOrden[i][1] = myBSRR;
00751     }
00752
00753     direccionRotacion = dir;
00754     return 0;
00755 }
00756
00762 int GestorSVM_Setaceleracion(int nuevaaceleracion) {
00763     if (flagChangingFrecuencia) {
00764         return -2;
00765     }
00766     if (nuevaaceleracion < ACELERACION_MINIMA || nuevaaceleracion > ACCLERACION_MAXIMA) {
00767         return -1;
00768     }
00769     aceleracion = nuevaaceleracion;
00770     return 0;
00771 }
00772
00778 int GestorSVM_SetDecel(int nuevaDecel) {
00779     if (flagChangingFrecuencia) {
00780         return -2;
00781     }
00782     if (nuevaDecel < DESACELERACION_MINIMA || nuevaDecel > DESACELERACION_MAXIMA) {
00783         return -1;
00784     }
00785     desaceleracion = nuevaDecel;
00786     return 0;
00787 }
00788
00794 int GestorSVM_MotorStart() {
00795     if (!flagMotorRunning) {
00796         flagMotorRunning = 1;
00797         flagEsAcelerado = 1;
00798         flagChangingFrecuencia = 1;
00799
00800         cambioFrecuenciaPorCiclo = (aceleracion * 1000 * 1000) / (frecuenciaSwitching);
00801
00802         /* Parámetros sombra de arranque */
00803         paramSombra.frecObjetivo = (uint32_t)frecuenciaReferenica * 1000 * 1000;
00804         paramSombra.flagMotorRunning = 1;
00805         paramSombra.flagEsAcelerado = 1;
00806         paramSombra.flagChangingFrecuencia = 1;
00807         paramSombra.cambioFrecuenciaPorCiclo = cambioFrecuenciaPorCiclo;
00808         flagActualizarParamSombra = 1;
00809
00810         /* Habilitar drivers */
```

```
00811         HAL_GPIO_WritePin(GPIOA, GPIO_U_SD, GPIO_PIN_SET);
00812         HAL_GPIO_WritePin(GPIOA, GPIO_V_SD, GPIO_PIN_SET);
00813         HAL_GPIO_WritePin(GPIOA, GPIO_W_SD, GPIO_PIN_SET);
00814 
00815         /* Precargar una muestra y sincronizar switching */
00816         GestorSVM_CalcInterrupt();
00817         GestorSVM_SwitchInterrupt(SWITCH_INT_RESET);
00818 
00819         /* Iniciar timer de switching */
00820         GestorTimers_IniciarTimerSVM();
00821         return 0;
00822     }
00823     return 1;
00824 }
00825 
00831 int GestorSVM_MotorStop() {
00832     if (flagMotorRunning) {
00833         flagEsAcelerado = 0;
00834         flagChangingFrecuencia = 1;
00835 
00836         cambioFrecuenciaPorCiclo = (desaceleracion * 1000 * 1000) / (frecuenciaSwitching);
00837         frecObjetivo = 0;
00838 
00839         /* Parámetros sombra */
00840         paramSombra.flagMotorRunning = 1;
00841         paramSombra.flagEsAcelerado = 0;
00842         paramSombra.flagChangingFrecuencia = 1;
00843         paramSombra.cambioFrecuenciaPorCiclo = cambioFrecuenciaPorCiclo;
00844         paramSombra.frecObjetivo = 0;
00845         flagActualizarParamSombra = 1;
00846 
00847         return 0;
00848     }
00849     return 1;
00850 }
00851 
00857 int GestorSVM_Estop() {
00858     if (flagMotorRunning) {
00859         GestorTimers_DetenerTimerSVM();
00860 
00861         HAL_GPIO_WritePin(GPIOA, GPIO_U_SD, GPIO_PIN_RESET);
00862         HAL_GPIO_WritePin(GPIOA, GPIO_V_SD, GPIO_PIN_RESET);
00863         HAL_GPIO_WritePin(GPIOA, GPIO_W_SD, GPIO_PIN_RESET);
00864 
00865         GestorSVM_SwitchInterrupt(SWITCH_INT_CLEAN);
00866 
00867         bufferCalculo.indiceEscritura = 0;
00868         bufferCalculo.indiceLectura   = 0;
00869         bufferCalculo.contadorDeDatos     = 0;
00870 
00871         HAL_GPIO_WritePin(GPIOA, GPIO_U_IN, GPIO_PIN_RESET);
00872         HAL_GPIO_WritePin(GPIOA, GPIO_V_IN, GPIO_PIN_RESET);
00873         HAL_GPIO_WritePin(GPIOA, GPIO_W_IN, GPIO_PIN_RESET);
00874 
00875         flagChangingFrecuencia = 0;
00876         flagMotorRunning = 0;
00877         frecuenciaSalida = 0;
00878     }
00879     return 0;
00880 }
00881 
00887 int GestorSVM_GetFrec() {
00888     return frecuenciaReferenica;
00889 }
00890 
00892 int GestorSVM_Getaceleracion() {
00893     return aceleracion;
00894 }
00895 
00897 int GestorSVM_GetDesaceleracion() {
00898     return desaceleracion;
00899 }
00900 
00902 int GestorSVM_GetDir() {
00903     return direccionRotacion;
00904 }
00905 
00906 /* ================================ ISR de TIM3 (HAL) ================================ */
00907 
00918 void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim) {
00919     if (htim->Instance != TIM3) {
00920         return;
00921     }
00922 
00923     switch (htim->Channel) {
00924         case HAL_TIM_ACTIVE_CHANNEL_1:
00925             GestorSVM_SwitchInterrupt(SWITCH_INT_RESET);
```

```
00926             break;
00927         case HAL_TIM_ACTIVE_CHANNEL_2:
00928             GestorSVM_SwitchInterrupt(SWITCH_INT_CH1);
00929             break;
00930         case HAL_TIM_ACTIVE_CHANNEL_3:
00931             GestorSVM_SwitchInterrupt(SWITCH_INT_CH2);
00932             break;
00933         case HAL_TIM_ACTIVE_CHANNEL_4:
00934             GestorSVM_SwitchInterrupt(SWITCH_INT_CH3);
00935             break;
00936         default:
00937             break;
00938     }
00939 }
```

9.13. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_SVM/GestorSVM.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Estructuras de datos

- struct ValoresSwitching

  Estructura de trabajo del ISR del timer de switching.
- struct ConfiguracionSVM

  Parámetros de configuración del módulo SVM.

defines

- #define FREC_SWITCH 2511
- #define FREC_REFERENCIA 50
- #define DIRECCION_ROTACION 1
- #define RESOLUCION_TIMER 255
- #define ACCELERACION_DEFAUL 5
- #define FERC_OUT_MIN 1
- #define FERC_OUT_MAX 150
- #define DESACELERACION_DEFAULT 3
- #define ACCLERACION_MAXIMA 50
- #define ACELERACION_MINIMA 1
- #define DESACELERACION_MAXIMA 50
- #define DESACELERACION_MINIMA 1

typedefs

- typedef struct ValoresSwitching ValoresSwitching
- typedef struct ConfiguracionSVM ConfiguracionSVM

Enumeraciones

- enum OrdenSwitch {
  ORDEN_SWITCH_1_UP = 0 , ORDEN_SWITCH_2_UP = 1 , ORDEN_SWITCH_3_UP = 2 , ORDEN_SWITCH_3_DOWN = 3 ,
  ORDEN_SWITCH_2_DOWN = 4 , ORDEN_SWITCH_1_DOWN = 5 }

  Ordenes simbólicas de conmutación por canal/flujo (UP/DOWN).
- enum SwitchInterruptType {
  SWITCH_INT_CH1 = 0 , SWITCH_INT_CH2 , SWITCH_INT_CH3 , SWITCH_INT_RESET ,
  SWITCH_INT_CLEAN }

  Fuentes de interrupción de switching usadas por el ISR.

Funciones

- void GestorSVM_Init ()

  Inicializa el gestor SVM y su estado interno.
- void GestorSVM_SetConfiguration (ConfiguracionSVM *configuracion)

  Carga la configuración base de SVM y prepara tablas de BSRR por cuadrante.
- int GestorSVM_MotorStart ()

  Arranca el motor: habilita drivers, inicia timers y rampa hacia frecuenciaReferenica.
- int GestorSVM_MotorStop ()

  Ordena frenado con rampa de desaceleracion hasta 0 Hz.
- int GestorSVM_Estop ()

  Parada de emergencia inmediata: desactiva timers, apaga drivers y salidas.
- int GestorSVM_SetFrec (int frec)

  Solicita nueva frecuencia objetivo (Hz). Inicia rampa si el motor está en marcha.
- int GestorSVM_SetDir (int dir)

  Cambia el sentido de giro si el motor está detenido.
- int GestorSVM_SetAcel (int acel)

  Actualiza la aceleración dinámica [Hz/seg].
- int GestorSVM_SetDecel (int decel)

  Actualiza desacelerada [Hz/s].
- int GestorSVM_GetFrec ()

  Devuelve la frecuencia de referencia configurada (Hz).
- int GestorSVM_GetAcel ()

  Obtiene la aceleración actual [Hz/seg].
- int GestorSVM_GetDesacel ()

  Obtiene la desaceleración actual [Hz/seg].
- int GestorSVM_GetDir ()

  Devuelve sentido de giro (1 horario, -1 antihorario).
- void GestorSVM_CalcInterrupt ()

  Handler llamado por el timer de cálculo (productor). Encola t1/t2/t3 si hay espacio.

9.13.1. Documentación de «define»

9.13.1.1. ACCELERACION_DEFAUL

```
#define ACCELERACION_DEFAUL 5
```

Definición en la línea 77 del archivo GestorSVM.h.

9.13.1.2. ACCLERACION_MAXIMA

```
#define ACCLERACION_MAXIMA 50
```

Definición en la línea 82 del archivo GestorSVM.h.

9.13.1.3. ACELERACION_MINIMA

```
#define ACELERACION_MINIMA 1
```

Definición en la línea 83 del archivo GestorSVM.h.

9.13.1.4. DESACELERACION_DEFAULT

```
#define DESACELERACION_DEFAULT 3
```

Definición en la línea 81 del archivo GestorSVM.h.

9.13.1.5. DESACELERACION_MAXIMA

```
#define DESACELERACION_MAXIMA 50
```

Definición en la línea 84 del archivo GestorSVM.h.

9.13.1.6. DESACELERACION_MINIMA

```
#define DESACELERACION_MINIMA 1
```

Definición en la línea 85 del archivo GestorSVM.h.

9.13.1.7. DIRECCION_ROTACION

```
#define DIRECCION_ROTACION 1
```

Definición en la línea 75 del archivo GestorSVM.h.

9.13.1.8. FERC_OUT_MAX

```
#define FERC_OUT_MAX 150
```

Definición en la línea 80 del archivo GestorSVM.h.

9.13.1.9. FERC_OUT_MIN

```
#define FERC_OUT_MIN 1
```

Definición en la línea 79 del archivo GestorSVM.h.

9.13.1.10. FREC_REFERENCIA

```
#define FREC_REFERENCIA 50
```

Definición en la línea 74 del archivo GestorSVM.h.

9.13.1.11. FREC_SWITCH

```
#define FREC_SWITCH 2511
```

Definición en la línea 73 del archivo GestorSVM.h.

9.13.1.12. RESOLUCION_TIMER

```
#define RESOLUCION_TIMER 255
```

Definición en la línea 76 del archivo GestorSVM.h.

9.13.2. Documentación de «typedef»

9.13.2.1. ConfiguracionSVM

```
typedef struct ConfiguracionSVM ConfiguracionSVM
```

9.13.2.2. ValoresSwitching

```
typedef struct ValoresSwitching ValoresSwitching
```

9.13.3. Documentación de enumeraciones

9.13.3.1. OrdenSwitch

```
enum OrdenSwitch
```

Ordenes simbólicas de conmutación por canal/flujo (UP/DOWN).

Mapear a eventos CCR (t1/t2/t3) y flancos (subida/bajada) facilita el código del ISR. El orden debe coincidir con la secuencia SVM configurada.

Valores de enumeraciones

ORDEN_SWITCH_1_UP	
ORDEN_SWITCH_2_UP	Evento t1 en flanco de subida.
ORDEN_SWITCH_3_UP	Evento t2 en flanco de subida.

ORDEN_SWITCH_3_DOWN	Evento t3 en flanco de subida.
ORDEN_SWITCH_2_DOWN	Evento t3 en flanco de bajada.
ORDEN_SWITCH_1_DOWN	Evento t2 en flanco de bajada.

Definición en la línea 46 del archivo GestorSVM.h.

9.13.3.2. SwitchInterruptType

```
enum SwitchInterruptType
```

Fuentes de interrupción de switching usadas por el ISR.

- CH1..CH3: comparaciones CCR activas.
- RESET: sincroniza y precarga nuevos ticks (cambio de sector).
- CLEAN: limpia flags/estado (p.ej. al detener motor).

Valores de enumeraciones

SWITCH_INT_CH1	
SWITCH_INT_CH2	Interrupción por CCR de t1.
SWITCH_INT_CH3	Interrupción por CCR de t2.
SWITCH_INT_RESET	Interrupción por CCR de t3.
SWITCH_INT_CLEAN	Re-armado de ticks/sincronización por ciclo.

Definición en la línea 63 del archivo GestorSVM.h.

9.13.4. Documentación de funciones

9.13.4.1. GestorSVM_CalcInterrupt()

```
void GestorSVM_CalcInterrupt (
            void )
```

Handler llamado por el timer de cálculo (productor). Encola t1/t2/t3 si hay espacio.

ISR (o handler llamado por ISR) del timer de cálculo.

Corre a 2× FREC_SWITCH (según tu diseño) para anticipar cómputos: si el buffer circular tiene espacio, calcula t1/t2/t3, cuadrante y posibles interferencias, y los encola para que el timer de switching (TIM3) los consuma. No bloquea si el buffer está lleno.

Nota

Tamaño de buffer: 3 entradas (pipeline productor/consumidor).

Definición en la línea 640 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.13.4.2. GestorSVM_Estop()

```
int GestorSVM_Estop (
            void )
```

Parada de emergencia inmediata: desactiva timers, apaga drivers y salidas.

Parada de emergencia inmediata.

Devuelve

Siempre 0.

Siempre 0.

Deshabilita timers/interrupts, apaga drivers y salidas sin rampa.

Atención

Uso ante condiciones de falla. Requiere nueva orden de arranque.

Definición en la línea 857 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.13.4.3. GestorSVM_GetAcel()

```
int GestorSVM_GetAcel ()
```

Obtiene la aceleración actual [Hz/seg].

Gráfico de llamadas a esta función:

9.13.4.4. GestorSVM_GetDesacel()

```
int GestorSVM_GetDesacel ()
```

Obtiene la desaceleración actual [Hz/seg].

Gráfico de llamadas a esta función:

9.13.4.5. GestorSVM_GetDir()

```
int GestorSVM_GetDir ()
```

Devuelve sentido de giro (1 horario, -1 antihorario).

Obtiene el sentido de giro actual (1 horario, -1 antihorario).

Definición en la línea 902 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.13.4.6. GestorSVM_GetFrec()

```
int GestorSVM_GetFrec (
            void )
```

Devuelve la frecuencia de referencia configurada (Hz).

Lee la frecuencia objetivo de referencia (no escalada).

Devuelve

frecuenciaReferenica.

Frecuencia de referencia [Hz].

Nota

Si querés la frecuencia *instantánea* en rampa, podrías exponer otra API.

Definición en la línea 887 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.13.4.7. GestorSVM_Init()

```
void GestorSVM_Init (
            void )
```

Inicializa el gestor SVM y su estado interno.

Pone a cero variables internas, limpia buffers de cálculo y deja flags en estado consistente (motor detenido, sin cambios pendientes).

9.13.4.8. GestorSVM_MotorStart()

```
int GestorSVM_MotorStart (
            void )
```

Arranca el motor: habilita drivers, inicia timers y rampa hacia frecuenciaReferenica.

Inicia la marcha: arranca rampa de aceleración hacia frecReferencia.

Devuelve

0 si arranca; 1 si ya estaba en marcha.

0 si inicia correctamente, 1 si ya estaba en marcha.

Habilita drivers, inicia timers (cálculo y switching), precarga ticks, y pone flags para rampa ascendente. La frecuencia objetivo proviene de GestorSVM_SetFrec o GestorSVM_SetConfiguration.

Definición en la línea 794 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.13.4.9. GestorSVM_MotorStop()

```
int GestorSVM_MotorStop (
            void )
```

Ordena frenado con rampa de desaceleracion hasta 0 Hz.

Ordena frenado controlado (rampa de desacel hasta 0 Hz).

Devuelve

0 si acepta; 1 si ya estaba detenido.

0 si se acepta la orden, 1 si ya estaba detenido.

No corta drivers instantáneamente: mantiene el switching hasta llegar a 0 Hz, luego apaga salidas y notifica al gestor de estados.

Definición en la línea 831 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.13.4.10. GestorSVM_SetAcel()

```
int GestorSVM_SetAcel (
            int acel)
```

Actualiza la aceleración dinámica [Hz/seg].

Parámetros

acel	Nueva aceleración.

Devuelve

0 OK; -1 fuera de rango; -2 si hay cambio de velocidad en curso.

Gráfico de llamadas a esta función:

9.13.4.11. GestorSVM_SetConfiguration()

```
void GestorSVM_SetConfiguration (
            ConfiguracionSVM * configuracion)
```

Carga la configuración base de SVM y prepara tablas de BSRR por cuadrante.

Carga la configuración base del SVM.

Parámetros

configuracion	Puntero a ConfiguracionSVM.
configuracion	Puntero a ConfiguracionSVM con frec_switch, frecReferencia, direccionRotacion, acel y desacel.

Aplica frec_switch, rampas y sentido; y propaga frecReferencia al pipeline de cambio de velocidad. Internamente recalcula lookups/BSRR por cuadrante.

Atención

Validar rangos antes de invocar si la config proviene de UI/externo.

Definición en la línea 585 del archivo GestorSVM.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.13.4.12. GestorSVM_SetDecel()

```
int GestorSVM_SetDecel (
            int nuevaDecel)
```

Actualiza desacelerada [Hz/s].

Actualiza la desaceleración dinámica [Hz/seg].

Devuelve

0 OK; -1 fuera de rango; -2 si hay rampa activa.

Parámetros

decel	Nueva desaceleración.

Devuelve

0 OK; -1 fuera de rango; -2 si hay cambio de velocidad en curso.

Definición en la línea 778 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.13.4.13. GestorSVM_SetDir()

```
int GestorSVM_SetDir (
            int dir)
```

Cambia el sentido de giro si el motor está detenido.

Cambia el sentido de giro (solo con motor detenido).

Parámetros

dir	0 o 1 (mapeo a antihorario/horario).

Devuelve

0 OK; -1 fuera de rango; -2 si motor en marcha o rampa activa.

Parámetros

dir	0 o 1 (mapea a antihorario/horario según tu implementación).

Devuelve

- 0: Modificado.
- -1: Fuera de rango.
- -2: Rechazado (motor en marcha o cambio en curso).

Recalcula la tabla BSRR por cuadrante para invertir U/V.

Definición en la línea 717 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.13.4.14. GestorSVM_SetFrec()

```
int GestorSVM_SetFrec (
            int frec)
```

Solicita nueva frecuencia objetivo (Hz). Inicia rampa si el motor está en marcha.

Solicita una nueva frecuencia objetivo.

Parámetros

frec	Frecuencia objetivo [Hz].

Devuelve

0 si aceptada (motor detenido), 1 si aceptada con rampa en curso; -1 fuera de rango; -2 misma que la actual.

Parámetros

frec	Frecuencia objetivo [Hz].

Devuelve

- 0: Aceptada con motor detenido (queda como referencia).
- 1: Aceptada con motor en marcha (inicia rampa).
- -1: Fuera de rango (FERC_OUT_MIN..FERC_OUT_MAX).
- -2: Igual a la frecuencia actual (sin cambios).

Calcula delta por ciclo a partir de acel/ desacel y frec_switch, y actualiza parámetros sombra para el lazo de cálculo.

Definición en la línea 668 del archivo GestorSVM.c.

Gráfico de llamadas a esta función:

9.14. GestorSVM.h

Ir a la documentación de este archivo.

```
00001 #ifndef GESTOR_SVM_GESTORSVM_H_
00002 #define GESTOR_SVM_GESTORSVM_H_
00003
00013 typedef struct ValoresSwitching {
00014     char cuadrante;
00015     char flagSwitch[3];
00016     int  ticks1[2];
00017     int  ticks2[2];
00018     int  ticks3[2];
00019     int  contador;
00020 } ValoresSwitching;
00021
00031 typedef struct ConfiguracionSVM {
00032     int frec_switch;
00033     int frecReferencia;
00034     int direccionRotacion;
00035     int acel;
00036     int desacel;
00037 } ConfiguracionSVM;
00038
00046 typedef enum {
00047     ORDEN_SWITCH_1_UP   = 0,
00048     ORDEN_SWITCH_2_UP   = 1,
00049     ORDEN_SWITCH_3_UP   = 2,
00050     ORDEN_SWITCH_3_DOWN = 3,
00051     ORDEN_SWITCH_2_DOWN = 4,
00052     ORDEN_SWITCH_1_DOWN = 5,
00053 } OrdenSwitch;
00054
00063 typedef enum {
00064     SWITCH_INT_CH1 = 0,
00065     SWITCH_INT_CH2,
00066     SWITCH_INT_CH3,
00067     SWITCH_INT_RESET,
00068     SWITCH_INT_CLEAN,
00069 } SwitchInterruptType;
00070
00071 /* -------------------- Parámetros de operación por defecto / límites -------------------- */
00072
00073 #define FREC_SWITCH                    2511
00074 #define FREC_REFERENCIA                50
00075 #define DIRECCION_ROTACION             1
00076 #define RESOLUCION_TIMER               255
00077 #define ACCELERACION_DEFAUL            5
00078
00079 #define FERC_OUT_MIN                   1
00080 #define FERC_OUT_MAX                   150
00081 #define DESACELERACION_DEFAULT         3
00082 #define ACCLERACION_MAXIMA             50
00083 #define ACELERACION_MINIMA             1
00084 #define DESACELERACION_MAXIMA          50
00085 #define DESACELERACION_MINIMA          1
00086
00094 void GestorSVM_Init();
00095
00106 void GestorSVM_SetConfiguration(ConfiguracionSVM* configuracion);
00107
00108 /* -------------------- API de control en tiempo de ejecución -------------------- */
00109
00119 int GestorSVM_MotorStart();
00120
00129 int GestorSVM_MotorStop();
00130
00139 int GestorSVM_Estop();
00140
00154 int GestorSVM_SetFrec(int frec);
00155
00167 int GestorSVM_SetDir(int dir);
00168
00175 int GestorSVM_SetAcel(int acel);
00176
00183 int GestorSVM_SetDecel(int decel);
00184
00191 int GestorSVM_GetFrec();
00192
00198 int GestorSVM_GetAcel();
00199
00204 int GestorSVM_GetDesacel();
00205
00211 int GestorSVM_GetDir();
00212
00223 void GestorSVM_CalcInterrupt();
```

```
00224
00225 #endif /* GESTOR_SVM_GESTORSVM_H_ */
```

9.15. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_Timers/GestorTimers.c

```
#include "../Gestor_Timers/GestorTimers.h"
#include "../Inc/main.h"
```

Gráfico de dependencias incluidas en GestorTimers.c:

Funciones

- void GestorTimers_Init (TIM_HandleTypeDef *_hTimerSwitch, TIM_HandleTypeDef *_hTimerCalc)
- void GestorTimers_IniciarTimerSVM ()
- void GestorTimers_DetenerTimerSVM ()

Variables

- TIM_HandleTypeDef * hTimerSwitch
- TIM_HandleTypeDef * hTimerCalc

9.15.1. Documentación de funciones

9.15.1.1. GestorTimers_DetenerTimerSVM()

```
void GestorTimers_DetenerTimerSVM ()
```

Definición en la línea 40 del archivo GestorTimers.c.

Gráfico de llamadas a esta función:

9.15.1.2. GestorTimers_IniciarTimerSVM()

```
void GestorTimers_IniciarTimerSVM ()
```

Definición en la línea 20 del archivo GestorTimers.c.

Gráfico de llamadas a esta función:

9.15.1.3. GestorTimers_Init()

```
void GestorTimers_Init (
            TIM_HandleTypeDef * _hTimerSwitch,
            TIM_HandleTypeDef * _hTimerCalc)
```

Definición en la línea 14 del archivo GestorTimers.c.

9.15.2. Documentación de variables

9.15.2.1. hTimerCalc

```
TIM_HandleTypeDef* hTimerCalc
```

Definición en la línea 12 del archivo GestorTimers.c.

9.15.2.2. hTimerSwitch

```
TIM_HandleTypeDef* hTimerSwitch
```

Definición en la línea 11 del archivo GestorTimers.c.

9.16. GestorTimers.c

Ir a la documentación de este archivo.

```
00001 #include "../Gestor_Timers/GestorTimers.h"
00002
00003 #include "../Inc/main.h"
00004
00005 // Este gestor va a mantener los timers. A este le vamos a pedir que nos inicie
00006 // y detenga los timers.
00007 // Los posibles timers ya estan establecidos
00008
00009
00010 // Dos punteros a las variables de timer
00011 TIM_HandleTypeDef* hTimerSwitch;
00012 TIM_HandleTypeDef* hTimerCalc;
00013
00014 void GestorTimers_Init(TIM_HandleTypeDef* _hTimerSwitch, TIM_HandleTypeDef* _hTimerCalc) {
00015     hTimerSwitch = _hTimerSwitch;
00016     hTimerCalc = _hTimerCalc;
00017 }
00018
00019
00020 void GestorTimers_IniciarTimerSVM() {
00021
00022     // Seteo de la prioridad del timer 2 a la segunda mas alta (1)
00023     HAL_NVIC_SetPriority(TIM2_IRQn, 1, 0);
00024     HAL_NVIC_EnableIRQ(TIM2_IRQn);
00025
00026     // Seteo de la prioridad del timer 3 a la prioridad mas alta (0)
00027     HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);
00028     HAL_NVIC_EnableIRQ(TIM3_IRQn);
00029
00030     // Se inician el timer 2
00031     HAL_TIM_IC_Start_IT(hTimerCalc, TIM_CHANNEL_1);
00032
00033     // Se inician los 4 canales del timer 3
00034     HAL_TIM_IC_Start_IT(hTimerSwitch, TIM_CHANNEL_1);
00035     HAL_TIM_IC_Start_IT(hTimerSwitch, TIM_CHANNEL_2);
00036     HAL_TIM_IC_Start_IT(hTimerSwitch, TIM_CHANNEL_3);
00037     HAL_TIM_IC_Start_IT(hTimerSwitch, TIM_CHANNEL_4);
00038 }
00039
00040 void GestorTimers_DetenerTimerSVM() {
00041
00042     // Se detiene el timer de switcheo
00043     HAL_TIM_IC_Stop_IT(hTimerSwitch, TIM_CHANNEL_1);
00044     HAL_TIM_IC_Stop_IT(hTimerSwitch, TIM_CHANNEL_2);
00045     HAL_TIM_IC_Stop_IT(hTimerSwitch, TIM_CHANNEL_3);
00046     HAL_TIM_IC_Stop_IT(hTimerSwitch, TIM_CHANNEL_4);
00047
00048     __HAL_TIM_SET_COUNTER(hTimerSwitch, 0);
00049
00050
00051     // Se detiene el timer de calculo
00052     HAL_TIM_IC_Stop_IT(hTimerCalc, TIM_CHANNEL_1);
00053     __HAL_TIM_SET_COUNTER(hTimerCalc, 0);
00054
00055 }
```

9.17. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/Gestor_Timers/GestorTimers.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Enumeraciones

- enum TimersEnum { SVM_Timer = 0 , Dinamic_Timer }

Funciones

- void GestorTimers_Init ()
- void GestorTimers_IniciarTimerSVM ()
- void GestorTimers_DetenerTimerSVM ()

9.17.1. Documentación de enumeraciones

9.17.1.1. TimersEnum

```
enum TimersEnum
```

Valores de enumeraciones

SVM_Timer	
Dinamic_Timer	

Definición en la línea 13 del archivo GestorTimers.h.

9.17.2. Documentación de funciones

9.17.2.1. GestorTimers_DetenerTimerSVM()

```
void GestorTimers_DetenerTimerSVM ()
```

Definición en la línea 40 del archivo GestorTimers.c.

Gráfico de llamadas a esta función:

9.17.2.2. GestorTimers_IniciarTimerSVM()

```
void GestorTimers_IniciarTimerSVM ()
```

Definición en la línea 20 del archivo GestorTimers.c.

Gráfico de llamadas a esta función:

9.17.2.3. GestorTimers_Init()

```
void GestorTimers_Init ()
```

Gráfico de llamadas a esta función:

9.18. GestorTimers.h

Ir a la documentación de este archivo.

```
00001 /*
00002  * GestorTimers.h
00003  *
00004  *  Created on: Aug 10, 2025
00005  *      Author: elian
00006  */
00007
00008 #ifndef GESTOR_TIMERS_GESTORTIMERS_H_
00009 #define GESTOR_TIMERS_GESTORTIMERS_H_
00010
00011
00012
00013 typedef enum {
00014     SVM_Timer = 0,
00015     Dinamic_Timer
00016 } TimersEnum;
00017
00018 void GestorTimers_Init();
00019
00020 void GestorTimers_IniciarTimerSVM();
00021 void GestorTimers_DetenerTimerSVM();
00022
00023 #endif /* GESTOR_TIMERS_GESTORTIMERS_H_ */
```

9.19. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/SPI_Interfase/SPIModule.c

```
#include <stdio.h>
#include <string.h>
#include "main.h"
#include "../Gestor_Estados/GestorEstados.h"
```

Gráfico de dependencias incluidas en SPIModule.c:

defines

- #define SPI_BUF_SIZE 16
- #define SPI_TRANSMITION_SIZE 4
- #define CMD_LEN 3

Funciones

- static void SPI_ProcesarComando (uint8_t *buffer, int cantBytes, uint8_t *bufferResponse)

  Procesa un comando recibido por SPI y construye la respuesta.
- void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef *_hspi)

  Callback de HAL llamado al completar una transacción TxRx por DMA.
- void SPI_Init (void *_hspi)

  Inicializa el módulo SPI esclavo con DMA y deja activa la primera transacción.

Variables

- uint8_t rxDMABuffer [SPI_BUF_SIZE]
- uint8_t txDMABuffer [SPI_BUF_SIZE]
- volatile uint8_t rxBuffer [SPI_BUF_SIZE]
- volatile int rxIndex = 0
- volatile uint8_t tx_buffer [CMD_LEN] = {0xAA, 0xBB, 0xCC}
- volatile uint8_t rx_index = 0
- SPI_HandleTypeDef * hspi

9.19.1. Documentación de «define»

9.19.1.1. CMD_LEN

```
#define CMD_LEN 3
```

Definición en la línea 16 del archivo SPIModule.c.

9.19.1.2. SPI_BUF_SIZE

```
#define SPI_BUF_SIZE 16
```

Definición en la línea 14 del archivo SPIModule.c.

9.19.1.3. SPI_TRANSMITION_SIZE

```
#define SPI_TRANSMITION_SIZE 4
```

Definición en la línea 15 del archivo SPIModule.c.

9.19.2. Documentación de funciones

9.19.2.1. HAL_SPI_TxRxCpltCallback()

```
void HAL_SPI_TxRxCpltCallback (
            SPI_HandleTypeDef * _hspi)
```

Callback de HAL llamado al completar una transacción TxRx por DMA.

- Busca el delimitador ';' en rxDMABuffer para conocer longitud de comando.
- Construye una respuesta por defecto (ERR_NO_COMMAND) si el frame es inválido.
- Llama a SPI_ProcesarComando() si hay un payload válido.
- Copia la respuesta a txDMABuffer para la PRÓXIMA transacción.
- Re-arma la transacción DMA continua con HAL_SPI_TransmitReceive_DMA().

Definición en la línea 223 del archivo SPIModule.c.

Gráfico de llamadas de esta función:

9.19.2.2. SPI_Init()

```
void SPI_Init (
            void * hspi)
```

Inicializa el módulo SPI esclavo con DMA y deja activa la primera transacción.

- Configura prioridades/enable de IRQ para DMA1 Channel 4/5 (Tx/Rx).
- Limpia buffers RX/TX y precarga una respuesta por defecto (`SPI_RESPONSE_OK;`).
- Llama a `HAL_SPI_TransmitReceive_DMA()` para iniciar el ciclo continuo DMA (el tamaño de paquete es el configurado en el fuente, p.ej. `SPI_TRANSMITION_SIZE`).
- El parseo y la construcción de SPI_Response se realizan en el callback `HAL_SPI_TxRxCpltCallback`, invocando GestorEstados_Action según corresponda.

Parámetros

in	*hspi*	Handler de SPI inicializado por HAL (tipo `SPI_HandleTypeDef*`, p.ej. `&hspi2`).

Ver también

GestorEstados_Action
SPI_Request
SPI_Response

Definición en la línea 257 del archivo SPIModule.c.

Gráfico de llamadas a esta función:

9.19.2.3. SPI_ProcesarComando()

```
void SPI_ProcesarComando (
            uint8_t * buffer,
            int cantBytes,
            uint8_t * bufferResponse)  [static]
```

Procesa un comando recibido por SPI y construye la respuesta.

Parámetros

buffer	Puntero al buffer recibido (sin el ';' final).
cantBytes	Cantidad de bytes válidos en buffer (antes de ';').
bufferResponse	Puntero a un arreglo destino (mín. 4 bytes) donde se escribe la respuesta en formato [RESP][';'][dato1][dato2].

Nota

La respuesta se deja en bufferResponse; el callback de DMA la copiará a txDMABuffer.

Para GET_FREC se empaquetan hasta 2 bytes (valor <= 511 según tu lógica actual).

Definición en la línea 43 del archivo SPIModule.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.19.3. Documentación de variables

9.19.3.1. hspi

```
SPI_HandleTypeDef* hspi
```

Definición en la línea 31 del archivo SPIModule.c.

9.19.3.2. rx_index

```
volatile uint8_t rx_index = 0
```

Definición en la línea 28 del archivo SPIModule.c.

9.19.3.3. rxBuffer

```
volatile uint8_t rxBuffer[SPI_BUF_SIZE]
```

Definición en la línea 23 del archivo SPIModule.c.

9.19.3.4. rxDMABuffer

```
uint8_t rxDMABuffer[SPI_BUF_SIZE]
```

Definición en la línea 19 del archivo SPIModule.c.

9.19.3.5. rxIndex

```
volatile int rxIndex = 0
```

Definición en la línea 24 del archivo SPIModule.c.

9.19.3.6. tx_buffer

```
volatile uint8_t tx_buffer[CMD_LEN] = {0xAA, 0xBB, 0xCC}
```

Definición en la línea 27 del archivo SPIModule.c.

9.19.3.7. txDMABuffer

```
uint8_t txDMABuffer[SPI_BUF_SIZE]
```

Definición en la línea 20 del archivo SPIModule.c.

9.20. SPIModule.c

Ir a la documentación de este archivo.

```
00001 /*
00002 *   SPIModule.c
00003 *
00004 *   Created on: 11 ago. 2025
00005 *       Author: elian
00006 */
00007
00008 #include <stdio.h>
00009 #include <string.h>
00010 #include "main.h"
00011 #include "../Gestor_Estados/GestorEstados.h"
00012
00013 /* Tamaños de buffers y frame SPI */
00014 #define SPI_BUF_SIZE          16   // Tamaño del buffer circular DMA RX/TX
00015 #define SPI_TRANSMITION_SIZE   4   // Longitud de transacción DMA (bytes)
00016 #define CMD_LEN                3   // Cantidad de bytes que envía el master en un comando típico
00017
00018 /* Buffers para DMA (HW lee/escribe aquí directamente) */
00019 uint8_t rxDMABuffer[SPI_BUF_SIZE];  // RX por DMA (lectura directa desde periférico)
00020 uint8_t txDMABuffer[SPI_BUF_SIZE];  // TX por DMA (próxima respuesta a enviar)
00021
00022 /* Buffers auxiliares opcionales (no usados con DMA continuo) */
00023 volatile uint8_t rxBuffer[SPI_BUF_SIZE];
00024 volatile int rxIndex = 0;
00025
00026 /* Respuesta inicial dummy (no usada en flujo actual, se mantiene por compatibilidad) */
00027 volatile uint8_t tx_buffer[CMD_LEN] = {0xAA, 0xBB, 0xCC};
00028 volatile uint8_t rx_index = 0;
00029
00030 /* Handle del periférico SPI (inyectado desde fuera con SPI_Init) */
00031 SPI_HandleTypeDef* hspi;
00032
00043 static void SPI_ProcesarComando(uint8_t* buffer, int cantBytes, uint8_t* bufferResponse) {
00044     int resp;
00045     int val;
00046
00047     // Selección por comando (primer byte del buffer de respuesta
00048     bufferResponse[0] = '\0';
00049     switch (buffer[0]) {
00050         case SPI_REQUEST_START:
00051             resp = GestorEstados_Action(ACTION_START, 0);
00052
00053             if(resp == ACTION_RESP_OK) {
00054                 bufferResponse[0] = SPI_RESPONSE_OK;
00055             } else if(resp == ACTION_RESP_MOVING) {
00056                 bufferResponse[0] = SPI_RESPONSE_ERR_MOVING;
00057             } else if(resp == ACTION_RESP_EMERGENCY_ACTIVE) {
00058                 bufferResponse[0] = SPI_RESPONSE_ERR_EMERGENCY_ACTIVE;
00059             } else {
00060                 bufferResponse[0] = SPI_RESPONSE_ERR;
00061             }
00062
00063             bufferResponse[1] = ';';
00064             return;
00065
00066         case SPI_REQUEST_STOP:
00067             resp = GestorEstados_Action(ACTION_STOP, 0);
00068
00069             if(resp == ACTION_RESP_OK) {
00070                 bufferResponse[0] = SPI_RESPONSE_OK;
00071             } else if(resp == ACTION_RESP_NOT_MOVING) {
00072                 bufferResponse[0] = SPI_RESPONSE_ERR_NOT_MOVING;
00073             } else {
00074                 bufferResponse[0] = SPI_RESPONSE_ERR;
00075             }
00076
00077             bufferResponse[1] = ';';
00078             return;
00079
00080         case SPI_REQUEST_EMERGENCY:
00081             GestorEstados_Action(ACTION_EMERGENCY, 0);
00082             bufferResponse[0] = SPI_RESPONSE_OK;
00083             bufferResponse[1] = ';';
00084             return;
00085
00086         case SPI_REQUEST_SET_FREC:
00087             val = (uint8_t)buffer[1] + (uint8_t)buffer[2];
00088             resp = GestorEstados_Action(ACTION_SET_FREC, val);
00089
00090             if(resp == ACTION_RESP_OK) {
00091                 bufferResponse[0] = SPI_RESPONSE_OK;
00092             } else if(resp == ACTION_RESP_OUT_RANGE) {
```

```
00093                 bufferResponse[0] = SPI_RESPONSE_ERR_DATA_OUT_RANGE;
00094             } else if(resp == ACTION_RESP_MOVING) {
00095                 bufferResponse[0] = SPI_RESPONSE_ERR_MOVING;
00096             } else {
00097                 bufferResponse[0] = SPI_RESPONSE_ERR;
00098             }
00099 
00100             bufferResponse[1] = ';';
00101             return;
00102 
00103         case SPI_REQUEST_SET_ACEL:
00104             resp = GestorEstados_Action(ACTION_SET_ACEL, (uint8_t)buffer[1]);
00105 
00106             if(resp == ACTION_RESP_OK) {
00107                 bufferResponse[0] = SPI_RESPONSE_OK;
00108             } else if(resp == ACTION_RESP_OUT_RANGE) {
00109                 bufferResponse[0] = SPI_RESPONSE_ERR_DATA_OUT_RANGE;
00110             } else if(resp == ACTION_RESP_MOVING) {
00111                 bufferResponse[0] = SPI_RESPONSE_ERR_MOVING;
00112             } else {
00113                 bufferResponse[0] = SPI_RESPONSE_ERR;
00114             }
00115 
00116             bufferResponse[1] = ';';
00117             return;
00118 
00119         case SPI_REQUEST_SET_DESACEL:
00120             resp = GestorEstados_Action(ACTION_SET_DESACEL, (uint8_t)buffer[1]);
00121 
00122             if(resp == ACTION_RESP_OK) {
00123                 bufferResponse[0] = SPI_RESPONSE_OK;
00124             } else if(resp == ACTION_RESP_OUT_RANGE) {
00125                 bufferResponse[0] = SPI_RESPONSE_ERR_DATA_OUT_RANGE;
00126             } else if(resp == ACTION_RESP_MOVING) {
00127                 bufferResponse[0] = SPI_RESPONSE_ERR_MOVING;
00128             } else {
00129                 bufferResponse[0] = SPI_RESPONSE_ERR;
00130             }
00131 
00132             bufferResponse[1] = ';';
00133             return;
00134 
00135         case SPI_REQUEST_SET_DIR:
00136             val = (uint8_t)buffer[1];
00137             resp = GestorEstados_Action(ACTION_SET_DIR, val);
00138 
00139             if(resp == ACTION_RESP_OK) {
00140                 bufferResponse[0] = SPI_RESPONSE_OK;
00141             } else if(resp == ACTION_RESP_OUT_RANGE) {
00142                 bufferResponse[0] = SPI_RESPONSE_ERR_DATA_OUT_RANGE;
00143             } else if(resp == ACTION_RESP_MOVING) {
00144                 bufferResponse[0] = SPI_RESPONSE_ERR_MOVING;
00145             } else {
00146                 bufferResponse[0] = SPI_RESPONSE_ERR;
00147             }
00148 
00149             bufferResponse[1] = ';';
00150             return;
00151 
00152         case SPI_REQUEST_GET_FREC:
00153             /* Lee valor actual de frecuencia desde el SVM */
00154             val = GestorSVM_GetFrec();
00155             bufferResponse[0] = SPI_RESPONSE_OK;
00156             /* Empaquetado simple en 2 bytes (LOW/HIGH "capado") */
00157             if (val > 255) {
00158                 bufferResponse[1] = 255;
00159                 bufferResponse[2] = (uint8_t)(val - 255);
00160             } else {
00161                 bufferResponse[1] = (uint8_t)val;
00162                 bufferResponse[2] = 0;
00163             }
00164             bufferResponse[3] = ';';
00165             return;
00166 
00167         case SPI_REQUEST_GET_ACEL:
00168             bufferResponse[0] = SPI_RESPONSE_OK;
00169             bufferResponse[1] = GestorSVM_GetAcel();
00170             bufferResponse[2] = 0;
00171             bufferResponse[3] = ';';
00172             return;
00173 
00174         case SPI_REQUEST_GET_DESACEL:
00175             bufferResponse[0] = SPI_RESPONSE_OK;
00176             bufferResponse[1] = GestorSVM_GetDesacel();
00177             bufferResponse[2] = 0;
00178             bufferResponse[3] = ';';
00179             return;
```

```
00180
00181         case SPI_REQUEST_GET_DIR:
00182             bufferResponse[0] = SPI_RESPONSE_OK;
00183             bufferResponse[1] = GestorSVM_GetDir();
00184             bufferResponse[2] = 0;
00185             bufferResponse[3] = ';';
00186             return;
00187
00188         case SPI_REQUEST_IS_STOP:
00189             bufferResponse[0] = SPI_RESPONSE_OK;
00190             bufferResponse[1] = (uint8_t)GestorEstados_Action(ACTION_IS_MOTOR_STOP, 0);
00191             bufferResponse[2] = 0;
00192             bufferResponse[3] = ';';
00193             return;
00194
00195         case SPI_REQUEST_RESPONSE:
00196             bufferResponse[0] = SPI_RESPONSE_OK;
00197             bufferResponse[1] = ';';
00198             return;
00199
00200         default:
00201             /* Comando desconocido */
00202             bufferResponse[0] = SPI_RESPONSE_ERR_CMD_UNKNOWN;
00203             bufferResponse[1] = ';';
00204             return;
00205     }
00206
00207     /* Fallback (no debería alcanzarse) */
00208     bufferResponse[0] = SPI_RESPONSE_ERR;
00209     bufferResponse[1] = ';';
00210 }
00211
00223 void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *_hspi) {
00224     int len = 0;
00225     int found = 0;
00226
00227     if (_hspi->Instance == SPI2) {
00228         /* Buscar fin de comando ';' en el buffer de RX */
00229         for (int i = 0; i < SPI_BUF_SIZE; i++) {
00230             if (rxDMABuffer[i] == ';') {
00231                 found = 1;
00232                 len = i;              // bytes válidos (sin incluir ';')
00233                 break;
00234             }
00235         }
00236
00237         /* Respuesta por defecto: "ERR_NO_COMMAND;" */
00238         uint8_t resp[4] = { SPI_RESPONSE_ERR_NO_COMMAND, ';', 0, 0 };
00239
00240         if (found && len > 0) {
00241             SPI_ProcesarComando(rxDMABuffer, len, resp);
00242         }
00243
00244         txDMABuffer[0] = resp[0];
00245         txDMABuffer[1] = resp[1];
00246         txDMABuffer[2] = resp[2];
00247         txDMABuffer[3] = resp[3];
00248
00249         /* Limpiar RX para próxima captura */
00250         memset(rxDMABuffer, 0, sizeof(rxDMABuffer));
00251
00252         /* Reiniciar ciclo DMA full-duplex (no bloqueante) */
00253         HAL_SPI_TransmitReceive_DMA(hspi, txDMABuffer, rxDMABuffer, SPI_TRANSMITION_SIZE);
00254     }
00255 }
00256
00257 void SPI_Init(void* _hspi) {
00258     /* Prioridades/enable de DMA para SPI2: TX (CH4), RX (CH5) */
00259     HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 2, 0);
00260     HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
00261
00262     HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 2, 0);
00263     HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);
00264
00265     /* Guardar handle de SPI */
00266     hspi = (SPI_HandleTypeDef*)_hspi;
00267
00268     /* Inicialización básica de buffers */
00269     memset(rxDMABuffer, 0, sizeof(rxDMABuffer));
00270     memset(txDMABuffer, 0, sizeof(txDMABuffer));
00271
00272     /* Respuesta inicial por defecto: "OK;" (se enviará en la primera TxRx) */
00273     txDMABuffer[0] = SPI_RESPONSE_OK;
00274     txDMABuffer[1] = ';';
00275
00276     /* Iniciar ciclo DMA full-duplex no bloqueante */
00277     HAL_SPI_TransmitReceive_DMA(hspi, txDMABuffer, rxDMABuffer, SPI_TRANSMITION_SIZE);
```

```
00278 }
```

9.21. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Modules/SPI_Interfase/SPIModule.h

Interfaz del módulo SPI (esclavo) con DMA.

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Enumeraciones

- enum SPI_Request {
  SPI_REQUEST_START = 10 , SPI_REQUEST_STOP , SPI_REQUEST_SET_FREC , SPI_REQUEST_SET_ACEL ,
  SPI_REQUEST_SET_DESACEL , SPI_REQUEST_SET_DIR , SPI_REQUEST_GET_FREC , SPI_REQUEST_GET_ACEL ,
  SPI_REQUEST_GET_DESACEL , SPI_REQUEST_GET_DIR , SPI_REQUEST_IS_STOP , SPI_REQUEST_EMERGENCY ,
  SPI_REQUEST_RESPONSE = 0x50 }

  Comandos aceptados por el esclavo a través de SPI.
- enum SPI_Response {
  SPI_RESPONSE_OK = 0xFF , SPI_RESPONSE_ERR = 0xA0 , SPI_RESPONSE_ERR_CMD_UNKNOWN , SPI_RESPONSE_ERR_NO_COMMAND ,
  SPI_RESPONSE_ERR_MOVING , SPI_RESPONSE_ERR_NOT_MOVING , SPI_RESPONSE_ERR_DATA_MISSING , SPI_RESPONSE_ERR_DATA_INVALID ,
  SPI_RESPONSE_ERR_DATA_OUT_RANGE , SPI_RESPONSE_ERR_EMERGENCY_ACTIVE , SPI_LAST_VALUE }

  Códigos de respuesta emitidos por el esclavo STM32.

Funciones

- void SPI_Init (void *hspi)

  Inicializa el módulo SPI esclavo con DMA y deja activa la primera transacción.

9.21.1. Descripción detallada

Interfaz del módulo SPI (esclavo) con DMA.

El maestro (ESP32) envía solicitudes SPI_Request y el STM32 responde con un código SPI_Response en la **siguiente** transacción DMA.
Este módulo actúa como capa de transporte sobre la lógica de estados implementada en GestorEstados_Action (ver SystemAction y SystemState).

Formato típico de tramas

- **Master** → **Slave**: `CMD [DATOS...] ';'` (ver SPI_Request)
- **Slave** → **Master**: `RESP [';'] [opcional: datos]` (ver SPI_Response)

Ver también

GestorEstados_Action
SystemAction
SystemState
SystemActionResponse

Definición en el archivo SPIModule.h.

9.21.2. Documentación de enumeraciones

9.21.2.1. SPI_Request

`enum SPI_Request`

Comandos aceptados por el esclavo a través de SPI.

Cada entrada se mapea a una acción de SystemAction procesada por GestorEstados_Action. Los comandos **SET_*** llevan datos; **GET_*** devuelven valores.

Valores de enumeraciones

SPI_REQUEST_START	
SPI_REQUEST_STOP	Solicita arranque → equivale a ACTION_START.
SPI_REQUEST_SET_FREC	Solicita parada → equivale a ACTION_STOP.
SPI_REQUEST_SET_ACEL	Setea frecuencia de régimen → ACTION_SET_FREC (requiere datos).
SPI_REQUEST_SET_DESACEL	Setea aceleración → ACTION_SET_ACEL (requiere datos).
SPI_REQUEST_SET_DIR	Setea desaceleración → ACTION_SET_DESACEL (requiere datos).
SPI_REQUEST_GET_FREC	Setea dirección de giro → ACTION_SET_DIR (requiere datos).
SPI_REQUEST_GET_ACEL	Consulta frecuencia actual → lectura sin cambio de SystemState.
SPI_REQUEST_GET_DESACEL	Consulta aceleración actual.
SPI_REQUEST_GET_DIR	Consulta desaceleración actual.
SPI_REQUEST_IS_STOP	Consulta dirección actual.
SPI_REQUEST_EMERGENCY	Consulta si el motor está detenido → usa ACTION_IS_MOTOR_STOP.
SPI_REQUEST_RESPONSE	Fuerza emergencia → ACTION_EMERGENCY.

Definición en la línea 30 del archivo SPIModule.h.

9.21.2.2. SPI_Response

`enum SPI_Response`

Códigos de respuesta emitidos por el esclavo STM32.

Conceptualmente mapean los resultados de GestorEstados_Action (ver SystemActionResponse) hacia el enlace SPI. En comandos **GET_*** pueden incluir bytes de datos adicionales.

Valores de enumeraciones

SPI_RESPONSE_OK	
SPI_RESPONSE_ERR	Operación válida/exitosa → similar a ACTION_RESP_OK.
SPI_RESPONSE_ERR_CMD_UNKNOWN	Error genérico → similar a ACTION_RESP_ERR.
SPI_RESPONSE_ERR_NO_COMMAND	Comando desconocido (no mapea a acción válida).
SPI_RESPONSE_ERR_MOVING	Trama sin comando o sin ';'.

SPI_RESPONSE_ERR_NOT_MOVING	Incompatible por movimiento → similar a ACTION_RESP_MOVING.
SPI_RESPONSE_ERR_DATA_MISSING	Incompatible por estar detenido → similar a ACTION_RESP_NOT_MOVING.
SPI_RESPONSE_ERR_DATA_INVALID	Faltan datos en un comando que los requiere.
SPI_RESPONSE_ERR_DATA_OUT_RANGE	Datos inválidos.
SPI_RESPONSE_ERR_EMERGENCY_ACTIVE	Datos fuera de rango → similar a ACTION_RESP_OUT_RANGE.
SPI_LAST_VALUE	No permitido en emergencia → similar a ACTION_RESP_EMERGENCY_ACTIVE. Marcador final (no usar como respuesta).

Definición en la línea 53 del archivo SPIModule.h.

9.21.3. Documentación de funciones

9.21.3.1. SPI_Init()

```
void SPI_Init (
            void * _hspi)
```

Inicializa el módulo SPI esclavo con DMA y deja activa la primera transacción.

- Configura prioridades/enable de IRQ para DMA1 Channel 4/5 (Tx/Rx).
- Limpia buffers RX/TX y precarga una respuesta por defecto (`SPI_RESPONSE_OK;`).
- Llama a `HAL_SPI_TransmitReceive_DMA()` para iniciar el ciclo continuo DMA (el tamaño de paquete es el configurado en el fuente, p.ej. `SPI_TRANSMITION_SIZE`).
- El parseo y la construcción de SPI_Response se realizan en el callback `HAL_SPI_TxRxCpltCallback`, invocando GestorEstados_Action según corresponda.

Parámetros

in	*hspi*	Handler de SPI inicializado por HAL (tipo `SPI_HandleTypeDef*`, p.ej. `&hspi2`).

Ver también

GestorEstados_Action

SPI_Request

SPI_Response

Definición en la línea 257 del archivo SPIModule.c.

Gráfico de llamadas a esta función:

9.22. SPIModule.h

Ir a la documentación de este archivo.

```
00001
00019
00020 #ifndef SPI_MODULE_H_
00021 #define SPI_MODULE_H_
00022
00030 typedef enum {
00031     SPI_REQUEST_START       = 10,
00032     SPI_REQUEST_STOP,
00033     SPI_REQUEST_SET_FREC,
00034     SPI_REQUEST_SET_ACEL,
00035     SPI_REQUEST_SET_DESACEL,
00036     SPI_REQUEST_SET_DIR,
00037     SPI_REQUEST_GET_FREC,
00038     SPI_REQUEST_GET_ACEL,
00039     SPI_REQUEST_GET_DESACEL,
00040     SPI_REQUEST_GET_DIR,
00041     SPI_REQUEST_IS_STOP,
00042     SPI_REQUEST_EMERGENCY,
00043     SPI_REQUEST_RESPONSE    = 0x50
00044 } SPI_Request;
00045
00053 typedef enum {
00054     SPI_RESPONSE_OK = 0xFF,
00055     SPI_RESPONSE_ERR = 0xA0,
00056     SPI_RESPONSE_ERR_CMD_UNKNOWN,
00057     SPI_RESPONSE_ERR_NO_COMMAND,
00058     SPI_RESPONSE_ERR_MOVING,
00059     SPI_RESPONSE_ERR_NOT_MOVING,
00060     SPI_RESPONSE_ERR_DATA_MISSING,
00061     SPI_RESPONSE_ERR_DATA_INVALID,
00062     SPI_RESPONSE_ERR_DATA_OUT_RANGE,
00063     SPI_RESPONSE_ERR_EMERGENCY_ACTIVE,
00064     SPI_LAST_VALUE
00065 } SPI_Response;
00066
00084 void SPI_Init(void* hspi);
00085
00086 #endif /* SPI_MODULE_H_ */
```

9.23. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src/main.c

Punto de entrada del firmware y configuración de periféricos.

```
#include "main.h"
#include <stdio.h>
#include "../Modules/Gestor_SVM/GestorSVM.h"
#include "../Modules/Gestor_Timers/GestorTimers.h"
#include "../Modules/Gestor_Estados/GestorEstados.h"
#include "../Modules/SPI_Interfase/SPIModule.h"
```

Gráfico de dependencias incluidas en main.c:

Funciones

- static void SystemClock_Config (void)

  Configura el clock del sistema a 72 MHz desde HSE=8 MHz con PLL×9.
- static void MX_GPIO_Init (void)

  Inicializa GPIOA y GPIOB.
- static void MX_DMA_Init (void)

  Habilita el clock del DMA1 y sus interrupciones de canal 4 y 5.

- static void MX_TIM3_Init (void)

  Función inicialización del timer 3 (SVM).
- static void MX_TIM2_Init (void)

  Función de inicialización del timer 2 (Pre cálculo).
- static void MX_USART1_UART_Init (void)

  Inicialización de la UART 1 (DEBUG).
- static void MX_SPI2_Init (void)

  SPI2 Initialization Function (HMI).
- int main (void)

  Función de entrada del sistema.
- void Error_Handler (void)

  Manejador global de errores fatales.

Variables

- SPI_HandleTypeDef hspi2
- DMA_HandleTypeDef hdma_spi2_tx
- DMA_HandleTypeDef hdma_spi2_rx
- TIM_HandleTypeDef htim2
- TIM_HandleTypeDef htim3
- UART_HandleTypeDef huart1

9.23.1. Descripción detallada

Punto de entrada del firmware y configuración de periféricos.

Inicializa reloj de sistema, GPIO, DMA, SPI2 (esclavo), USART1 (debug), y TIM2/TIM3 (cálculo y switching). Integra módulos GestorSVM, GestorTimers, GestorEstados y SPI module. Disposicion de transistores, los transistores son los Q1, Q2, Q3, Q4, Q5 y Q6 | | | Q1 Q2 Q3 | | | | | | Q4 Q5 Q6 | | | Como esto esta controlado por drivers de mosfets, entonces vamos a poner un 1 para activar el transistor superior y un cero para el transistor inferior. Vamos a tener ocho vectores en que vamos a trabajar: v0, v1, v2, v3, v4, v5, v6 y v7. Siendo el v0 y v7 dos vectores nulos. Cada uno de estos cuadrantes tiene un valor que indica el valor de los transistores:

- v0 = 000,
- v1 = 001,
- v2 = 010,
- v3 = 011,
- v4 = 100,
- v5 = 101,
- v6 = 110,
- v7 = 111. Ahora vamos a detallar por cuadrante como nos va a quedar y que cambia en cada cuadrante

- Quad 1: v0, v1, v3, v7 => P3: High, P2: High, P1: High, P1: Low, P2: Low, P3: Low
- Quad 2: v0, v2, v3, v7 => P2: High, P3: High, P1: High, P1: Low, P3: Low, P2: Low
- Quad 3: v0, v2, v6, v7 => P2: High, P1: High, P3: High, P3: Low, P1: Low, P2: Low
- Quad 4: v0, v4, v6, v7 => P1: High, P2: High, P3: High, P3: Low, P2: Low, P1: Low
- Quad 5: v0, v4, v5, v7 => P1: High, P3: High, P2: High, P2: Low, P3: Low, P1: Low
- Quad 6: v0, v1, v5, v7 => P3: High, P1: High, P2: High, P2: Low, P1: Low, P3: Low

Definición en el archivo main.c.

9.23.2. Documentación de funciones

9.23.2.1. Error_Handler()

```
void Error_Handler (
            void )
```

Manejador global de errores fatales.

Manejador genérico de errores.

Detiene la ejecución en un bucle y puede ser utilizado para reportar/diagnosticar condiciones críticas. Implementado en main.c.

Deshabilita IRQs y entra en bucle infinito para provocar reset por IWDG si está activo.

Definición en la línea 375 del archivo main.c.

Gráfico de llamadas a esta función:

9.23.2.2. main()

```
int main (
            void )
```

Función de entrada del sistema.

La función main inicializa la estructura de configuración, inicializa los periféricos y el gestor de estados. Al terminar su trabajo queda en un while(1) con la sentencia WFI para reducir el consumo de CPU. 1) HAL_Init() 2) SystemClock_Config() 3) Carga configuración SVM (frecuencia de switching, ref, etc.). 4) Inicializa manejador de timers (GestorTimers_Init). 5) Inicializa periféricos (GPIO, DMA, TIM3, USART1, TIM2, SPI2) y driver SPI. 6) Notifica fin de init al Gestor de Estados (ACTION_INIT_DONE). 7) Entra en lazo con WFI para ahorrar CPU, atendiendo a interrupciones.

Valores devueltos

int	Sin uso

Configuración del MCU. Reset of all peripherals, Initializes the Flash interface and the Systick.

Cofiguración del módulo SVM

Frecuencia de 2.511kHz para el switch del timer 3 entre subida y bajada del timer 3

Frecuencia de salida

Sentido horario

Aceleracón del sistema por default [Hz/seg]

Desaceleracón del sistema por default [Hz/seg]

Definición en la línea 132 del archivo main.c.

Gráfico de llamadas de esta función:

9.23.2.3. MX_DMA_Init()

```
void MX_DMA_Init (
            void )  [static]
```

Habilita el clock del DMA1 y sus interrupciones de canal 4 y 5.

Habilita el reloj del **DMA1** y registra en el NVIC las IRQ de **DMA1_Channel4** y **DMA1_Channel5** con prioridad 2. En F1, estos canales suelen mapearse a **SPI2_RX (Ch4)** y **SPI2_TX (Ch5)**, por lo que esta rutina deja listo el DMA para que llamadas como `HAL_SPI_TransmitReceive_DMA()` configuren y lancen las transferencias.
No configura direcciones, tamaños ni modos del DMA; solo habilita el clock y las IRQ.

Nota

Anteriormente la prioridad era 0. Luego pasó a 2 ya que en el SPI se configura en 2.

Definición en la línea 337 del archivo main.c.

Gráfico de llamadas a esta función:

9.23.2.4. MX_GPIO_Init()

```
void MX_GPIO_Init (
            void )  [static]
```

Inicializa GPIOA y GPIOB.

Configura los pines de los puertos A y B como salidas push pull, sin pull up y como pines de baja frecuencia. Los pines configurados son los que controlan el modulador y salidas (leds de estado). Las entradas digitales (termoswitch y boton de stop) no son configuradas ya que serán señales controladas por el HMI.

Definición en la línea 345 del archivo main.c.

Gráfico de llamadas a esta función:

9.23.2.5. MX_SPI2_Init()

```
void MX_SPI2_Init (
            void )  [static]
```

SPI2 Initialization Function (HMI).

Configura al SPI2 como esclavo, dependiendo de la comuincación con el maestro, el ESP32 para poder transaccionar información. Inicializa un tamaño de 8 bits de datos, con dos líneas de comunicación (MISO y MOSI) comenzando el byte por bit más gignificativo. No utiliza el módulo CRC, el latch del dato se toma en el flanco decreciente del clock y la línea en reposo queda en estado alto. El pin de selección de esclavo es utilizado por el maestro para iniciar la comunicación.

Definición en la línea 208 del archivo main.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.23.2.6. MX_TIM2_Init()

```
void MX_TIM2_Init (
            void )  [static]
```

Función de inicialización del timer 2 (Pre cálculo).

El timer funcionará a 10KHz tratando de adelantar los cálculos del timer 3, permitiendo agilizar la carga de sus contadores y evitando que haya problemas en la señal de salida.

Definición en la línea 225 del archivo main.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.23.2.7. MX_TIM3_Init()

```
void MX_TIM3_Init (
            void )  [static]
```

Función inicialización del timer 3 (SVM).

El timer 3 es utilizado para ejecutar el swtching de los pines del variador de frecuencia. Trabaja con un timer de 0 a 256 utilizando el clock interno como fuente de conteo dando un período de 398.22us (f = 2511Hz) y 4 canales de interrupción CCR1-4 que se irán corriendo conforme vaya avanzando la señal y cambiando la tensión de salida buscada de acuerdo a lo que dicte el estado del módulo SVM.

Definición en la línea 264 del archivo main.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.23.2.8. MX_USART1_UART_Init()

```
void MX_USART1_UART_Init (
            void )  [static]
```

Inicialización de la UART 1 (DEBUG).

El periférico se utilizará como puerto de debug. Solo tendrá pines de transmisión y recepción sin control de flujo; con configuración 115200,8,N,1

Definición en la línea 323 del archivo main.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.23.2.9. SystemClock_Config()

```
void SystemClock_Config (
            void )  [static]
```

Configura el clock del sistema a 72 MHz desde HSE=8 MHz con PLL×9.

Activa HSE, configura PLL (source=HSE, mul=×9) y selecciona SYSCLK=PLL. AHB=72 MHz, APB2=72 MHz (div1), APB1=36 MHz (div2), FLASH_LATENCY=2.

Definición en la línea 180 del archivo main.c.

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

9.23.3. Documentación de variables

9.23.3.1. hdma_spi2_rx

```
DMA_HandleTypeDef hdma_spi2_rx
```

Definición en la línea 44 del archivo main.c.

9.23.3.2. hdma_spi2_tx

```
DMA_HandleTypeDef hdma_spi2_tx
```

Definición en la línea 43 del archivo main.c.

9.23.3.3. hspi2

```
SPI_HandleTypeDef hspi2
```

Definición en la línea 42 del archivo main.c.

9.23.3.4. htim2

```
TIM_HandleTypeDef htim2
```

Definición en la línea 45 del archivo main.c.

9.23.3.5. htim3

```
TIM_HandleTypeDef htim3
```

Definición en la línea 46 del archivo main.c.

9.23.3.6. huart1

```
UART_HandleTypeDef huart1
```

Definición en la línea 47 del archivo main.c.

9.24. main.c

Ir a la documentación de este archivo.

```
00001
00033
00034 #include "main.h"
00035 #include <stdio.h>
00036
00037 #include "../Modules/Gestor_SVM/GestorSVM.h"
00038 #include "../Modules/Gestor_Timers/GestorTimers.h"
00039 #include "../Modules/Gestor_Estados/GestorEstados.h"
00040 #include "../Modules/SPI_Interfase/SPIModule.h"
00041
00042 SPI_HandleTypeDef hspi2;
00043 DMA_HandleTypeDef hdma_spi2_tx;
00044 DMA_HandleTypeDef hdma_spi2_rx;
00045 TIM_HandleTypeDef htim2;
00046 TIM_HandleTypeDef htim3;
00047 UART_HandleTypeDef huart1;
00048
00055  static void SystemClock_Config(void);
00056
00064 static void MX_GPIO_Init(void);
00065
00076 static void MX_DMA_Init(void);
00077
00085 static void MX_TIM3_Init(void);
00086
00094 static void MX_TIM2_Init(void);
00095
00103 static void MX_USART1_UART_Init(void);
00104
00112 static void MX_SPI2_Init(void);
00113
00132 int main(void) {
00133
00134   HAL_Init();
00135   SystemClock_Config();
00136
00137
00138   ConfiguracionSVM config = {
00139     .frec_switch = 2511,
00140     .frecReferencia = 50,
00141     .direccionRotacion = 1,
00142     .acel = 5,
00143     .desacel = 3,
00144   };
00145
00146   // Cargamos los numeros de los pines
00147   config.puerto_senal_pierna[0] = GPIO_PIN_1;
00148   config.puerto_senal_pierna[1] = GPIO_PIN_3;
00149   config.puerto_senal_pierna[2] = GPIO_PIN_5;
00150   config.puerto_encen_pierna[0] = GPIO_PIN_2;
00151   config.puerto_encen_pierna[1] = GPIO_PIN_4;
00152   config.puerto_encen_pierna[2] = GPIO_PIN_6;
00153   GestorSVM_SetConfiguration(&config);
00154
00155   // Initialize all configured peripherals
00156   MX_GPIO_Init();
00157   MX_DMA_Init();
00158   MX_TIM3_Init();
00159   MX_TIM2_Init();
00160   MX_USART1_UART_Init();
00161   MX_SPI2_Init();
00162   SPI_Init(&hspi2);
00163
00164   // Inicio del gestor de timers
00165   GestorTimers_Init(&htim3, &htim2);
00166
00167   // Informamos al gestor de estados que finalizo la inicializacion
00168   // Esta debe ser la ultima llama de funcion del init
00169   GestorEstados_Action(ACTION_INIT_DONE, 0);
00170   printf("Config done\n");
00171
00172   __HAL_DBGMCU_FREEZE_TIM3();
00173   __HAL_DBGMCU_FREEZE_TIM2();
00174
00175   while (1) {
00176     __WFI();
00177   }
00178 }
00179
00180 static void SystemClock_Config(void) {
00181   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
00182   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
00183
00184   // Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef
      structure.
00185   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00186   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00187   RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
00188   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00189   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00190   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00191   RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
00192   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
00193     Error_Handler();
00194   }
00195
00196   // Initializes the CPU, AHB and APB buses clocks
00197   RCC_ClkInitStruct.ClockType =
      RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
00198   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00199   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00200   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
00201   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00202
00203   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) {
00204     Error_Handler();
00205   }
00206 }
00207
00208 static void MX_SPI2_Init(void) {
00209   hspi2.Instance = SPI2;
00210   hspi2.Init.Mode = SPI_MODE_SLAVE;
00211   hspi2.Init.Direction = SPI_DIRECTION_2LINES;
00212   hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
00213   hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
00214   hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
00215   hspi2.Init.NSS = SPI_NSS_HARD_INPUT;
00216   hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
00217   hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
00218   hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
00219   hspi2.Init.CRCPolynomial = 10;
00220   if (HAL_SPI_Init(&hspi2) != HAL_OK) {
00221     Error_Handler();
00222   }
00223 }
00224
00225 static void MX_TIM2_Init(void) {
00226
00227   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
00228   TIM_MasterConfigTypeDef sMasterConfig = {0};
00229   TIM_OC_InitTypeDef sConfigOC = {0};
00230
00231   htim2.Instance = TIM2;
00232   htim2.Init.Prescaler = 0;
00233   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
00234   htim2.Init.Period = 7199;
00235   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00236   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00237   if (HAL_TIM_Base_Init(&htim2) != HAL_OK) {
00238     Error_Handler();
00239   }
00240
00241   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
00242   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK) {
00243     Error_Handler();
00244   }
00245   if (HAL_TIM_OC_Init(&htim2) != HAL_OK) {
00246     Error_Handler();
00247   }
00248
00249   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00250   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00251   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK) {
00252     Error_Handler();
00253   }
00254
00255   sConfigOC.OCMode = TIM_OCMODE_TIMING;
00256   sConfigOC.Pulse = 5;
00257   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
00258   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00259   if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK) {
00260     Error_Handler();
00261   }
00262 }
00263
00264 static void MX_TIM3_Init(void) {
00265   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
00266   TIM_MasterConfigTypeDef sMasterConfig = {0};
00267   TIM_OC_InitTypeDef sConfigOC = {0};
```

```
00268
00269   htim3.Instance = TIM3;
00270   htim3.Init.Prescaler = 55;
00271   htim3.Init.CounterMode = TIM_COUNTERMODE_CENTERALIGNED3;
00272   htim3.Init.Period = 256;
00273   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00274   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
00275   if (HAL_TIM_Base_Init(&htim3) != HAL_OK) {
00276     Error_Handler();
00277   }
00278
00279   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
00280   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK) {
00281     Error_Handler();
00282   }
00283   if (HAL_TIM_OC_Init(&htim3) != HAL_OK) {
00284     Error_Handler();
00285   }
00286
00287   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00288   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00289   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK) {
00290     Error_Handler();
00291   }
00292
00293   sConfigOC.OCMode = TIM_OCMODE_TIMING;
00294   sConfigOC.Pulse = 0;
00295   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
00296   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00297   if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
00298   {
00299     Error_Handler();
00300   }
00301   __HAL_TIM_ENABLE_OCxPRELOAD(&htim3, TIM_CHANNEL_1);
00302   sConfigOC.Pulse = 20;
00303   if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
00304   {
00305     Error_Handler();
00306   }
00307   __HAL_TIM_ENABLE_OCxPRELOAD(&htim3, TIM_CHANNEL_2);
00308   sConfigOC.Pulse = 80;
00309   if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
00310   {
00311     Error_Handler();
00312   }
00313   __HAL_TIM_ENABLE_OCxPRELOAD(&htim3, TIM_CHANNEL_3);
00314   sConfigOC.Pulse = 240;
00315   if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
00316   {
00317     Error_Handler();
00318   }
00319   __HAL_TIM_ENABLE_OCxPRELOAD(&htim3, TIM_CHANNEL_4);
00320
00321 }
00322
00323 static void MX_USART1_UART_Init(void) {
00324   huart1.Instance = USART1;
00325   huart1.Init.BaudRate = 115200;
00326   huart1.Init.WordLength = UART_WORDLENGTH_8B;
00327   huart1.Init.StopBits = UART_STOPBITS_1;
00328   huart1.Init.Parity = UART_PARITY_NONE;
00329   huart1.Init.Mode = UART_MODE_TX_RX;
00330   huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
00331   huart1.Init.OverSampling = UART_OVERSAMPLING_16;
00332   if (HAL_UART_Init(&huart1) != HAL_OK) {
00333     Error_Handler();
00334   }
00335 }
00336
00337 static void MX_DMA_Init(void) {
00338   __HAL_RCC_DMA1_CLK_ENABLE();
00339   HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 2, 0);
00340   HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
00341   HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 2, 0);
00342   HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);
00343 }
00344
00345 static void MX_GPIO_Init(void) {
00346   GPIO_InitTypeDef GPIO_InitStruct = {0};
00347
00348   /* GPIO Ports Clock Enable */
00349   __HAL_RCC_GPIOD_CLK_ENABLE();
00350   __HAL_RCC_GPIOA_CLK_ENABLE();
00351   __HAL_RCC_GPIOB_CLK_ENABLE();
00352
00353   HAL_GPIO_WritePin(GPIOA, GPIO_U_IN|GPIO_U_SD|GPIO_V_IN|GPIO_V_SD|GPIO_W_IN|GPIO_W_SD,
      GPIO_PIN_RESET);//|GPIO_TERMO_SWITCH|GPIO_STOP_BUTTON, GPIO_PIN_RESET);
```

```
00354   HAL_GPIO_WritePin(GPIOB, GPIO_LED_STATE|GPIO_LED_ERROR, GPIO_PIN_RESET);
00355
00356   GPIO_InitStruct.Pin =
      GPIO_U_IN|GPIO_U_SD|GPIO_V_IN|GPIO_V_SD|GPIO_W_IN|GPIO_W_SD|GPIO_TERMO_SWITCH|GPIO_STOP_BUTTON;
00357   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00358   GPIO_InitStruct.Pull = GPIO_NOPULL;
00359   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00360   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00361
00362   GPIO_InitStruct.Pin = GPIO_LED_STATE|GPIO_LED_ERROR;
00363   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00364   GPIO_InitStruct.Pull = GPIO_NOPULL;
00365   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00366   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00367 }
00368
00375 void Error_Handler(void) {
00376   __disable_irq();
00377   while (1) {
00378   }
00379 }
00380
00381 #ifdef  USE_FULL_ASSERT
00388 void assert_failed(uint8_t *file, uint32_t line);
00389 void assert_failed(uint8_t *file, uint32_t line) {
00390   printf("Wrong parameters value: file %s on line %d\r\n", file, line)
00391 }
00392 #endif /* USE_FULL_ASSERT */
```

9.25. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src/stm32f1xx_hal_msp.c

This file provides code for the MSP Initialization and de-Initialization codes.

`#include "main.h"`
Gráfico de dependencias incluidas en stm32f1xx_hal_msp.c:

Funciones

- void HAL_MspInit (void)
- void HAL_SPI_MspInit (SPI_HandleTypeDef *hspi)

  SPI MSP Initialization This function configures the hardware resources used in this example.
- void HAL_SPI_MspDeInit (SPI_HandleTypeDef *hspi)

  SPI MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_TIM_Base_MspInit (TIM_HandleTypeDef *htim_base)

  TIM_Base MSP Initialization This function configures the hardware resources used in this example.
- void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef *htim_base)

  TIM_Base MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_UART_MspInit (UART_HandleTypeDef *huart)

  UART MSP Initialization This function configures the hardware resources used in this example.
- void HAL_UART_MspDeInit (UART_HandleTypeDef *huart)

  UART MSP De-Initialization This function freeze the hardware resources used in this example.

Variables

- DMA_HandleTypeDef hdma_spi2_tx
- DMA_HandleTypeDef hdma_spi2_rx

9.25.1. Descripción detallada

This file provides code for the MSP Initialization and de-Initialization codes.

Atención

Definición en el archivo stm32f1xx_hal_msp.c.

9.25.2. Documentación de funciones

9.25.2.1. HAL_MspInit()

```
void HAL_MspInit (
             void )
```

Initializes the Global MSP.

Definición en la línea 27 del archivo stm32f1xx_hal_msp.c.

9.25.2.2. HAL_SPI_MspDeInit()

```
void HAL_SPI_MspDeInit (
             SPI_HandleTypeDef * hspi)
```

SPI MSP De-Initialization This function freeze the hardware resources used in this example.

Parámetros

hspi	SPI handle pointer

Valores devueltos

None	

SPI2 GPIO Configuration PB12 ------> SPI2_NSS PB13 ------> SPI2_SCK PB14 ------> SPI2_MISO PB15 ------> SPI2_MOSI

Definición en la línea 101 del archivo stm32f1xx_hal_msp.c.

9.25.2.3. HAL_SPI_MspInit()

```
void HAL_SPI_MspInit (
             SPI_HandleTypeDef * hspi)
```

SPI MSP Initialization This function configures the hardware resources used in this example.

Parámetros

hspi	SPI handle pointer

Valores devueltos

None	

SPI2 GPIO Configuration PB12 ------> SPI2_NSS PB13 ------> SPI2_SCK PB14 ------> SPI2_MISO PB15 ------> SPI2_MOSI

Definición en la línea 39 del archivo stm32f1xx_hal_msp.c.

Gráfico de llamadas de esta función:

9.25.2.4. HAL_TIM_Base_MspDeInit()

```
void HAL_TIM_Base_MspDeInit (
            TIM_HandleTypeDef * htim_base)
```

TIM_Base MSP De-Initialization This function freeze the hardware resources used in this example.

Parámetros

htim_base	TIM_Base handle pointer

Valores devueltos

None	

Definición en la línea 144 del archivo stm32f1xx_hal_msp.c.

9.25.2.5. HAL_TIM_Base_MspInit()

```
void HAL_TIM_Base_MspInit (
            TIM_HandleTypeDef * htim_base)
```

TIM_Base MSP Initialization This function configures the hardware resources used in this example.

Parámetros

htim_base	TIM_Base handle pointer

Valores devueltos

None	

Definición en la línea 126 del archivo stm32f1xx_hal_msp.c.

9.25.2.6. HAL_UART_MspDeInit()

```
void HAL_UART_MspDeInit (
            UART_HandleTypeDef * huart)
```

UART MSP De-Initialization This function freeze the hardware resources used in this example.

Parámetros

huart	UART handle pointer

Valores devueltos

None	

Definición en la línea 187 del archivo stm32f1xx_hal_msp.c.

9.25.2.7. HAL_UART_MspInit()

```
void HAL_UART_MspInit (
            UART_HandleTypeDef * huart)
```

UART MSP Initialization This function configures the hardware resources used in this example.

Parámetros

huart	UART handle pointer

Valores devueltos

None	

Definición en la línea 160 del archivo stm32f1xx_hal_msp.c.

9.25.3. Documentación de variables

9.25.3.1. hdma_spi2_rx

```
DMA_HandleTypeDef hdma_spi2_rx  [extern]
```

Definición en la línea 44 del archivo main.c.

9.25.3.2. hdma_spi2_tx

```
DMA_HandleTypeDef hdma_spi2_tx  [extern]
```

Definición en la línea 43 del archivo main.c.

9.26. stm32f1xx_hal_msp.c

Ir a la documentación de este archivo.

```
00001
00018
00019 #include "main.h"
00020
00021 extern DMA_HandleTypeDef hdma_spi2_tx;
00022 extern DMA_HandleTypeDef hdma_spi2_rx;
00023
00027 void HAL_MspInit(void) {
00028   __HAL_RCC_AFIO_CLK_ENABLE();
00029   __HAL_RCC_PWR_CLK_ENABLE();
00030   __HAL_AFIO_REMAP_SWJ_NOJTAG();
00031 }
00032
00039 void HAL_SPI_MspInit(SPI_HandleTypeDef* hspi) {
00040   GPIO_InitTypeDef GPIO_InitStruct = {0};
00041   if(hspi->Instance==SPI2) {
00042     __HAL_RCC_SPI2_CLK_ENABLE();
00043
00044     __HAL_RCC_GPIOB_CLK_ENABLE();
00051     GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_15;
00052     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00053     GPIO_InitStruct.Pull = GPIO_NOPULL;
00054     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00055
00056     GPIO_InitStruct.Pin = GPIO_PIN_14;
00057     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00058     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
00059     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00060
00061     hdma_spi2_tx.Instance = DMA1_Channel5;
00062     hdma_spi2_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
00063     hdma_spi2_tx.Init.PeriphInc = DMA_PINC_DISABLE;
00064     hdma_spi2_tx.Init.MemInc = DMA_MINC_ENABLE;
00065     hdma_spi2_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
00066     hdma_spi2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
00067     hdma_spi2_tx.Init.Mode = DMA_NORMAL;
00068     hdma_spi2_tx.Init.Priority = DMA_PRIORITY_LOW;
00069     if (HAL_DMA_Init(&hdma_spi2_tx) != HAL_OK) {
00070       Error_Handler();
00071     }
00072
00073     __HAL_LINKDMA(hspi,hdmatx,hdma_spi2_tx);
00074
00075     hdma_spi2_rx.Instance = DMA1_Channel4;
00076     hdma_spi2_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
00077     hdma_spi2_rx.Init.PeriphInc = DMA_PINC_DISABLE;
00078     hdma_spi2_rx.Init.MemInc = DMA_MINC_ENABLE;
00079     hdma_spi2_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
00080     hdma_spi2_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
00081     hdma_spi2_rx.Init.Mode = DMA_NORMAL;
00082     hdma_spi2_rx.Init.Priority = DMA_PRIORITY_LOW;
00083     if (HAL_DMA_Init(&hdma_spi2_rx) != HAL_OK) {
00084       Error_Handler();
00085     }
00086
00087     __HAL_LINKDMA(hspi,hdmarx,hdma_spi2_rx);
00088
00089     HAL_NVIC_SetPriority(SPI2_IRQn, 0, 0);
00090     HAL_NVIC_EnableIRQ(SPI2_IRQn);
00091   }
00092
00093 }
00094
00101 void HAL_SPI_MspDeInit(SPI_HandleTypeDef* hspi) {
00102   if(hspi->Instance==SPI2) {
00103     __HAL_RCC_SPI2_CLK_DISABLE();
00104
00111     HAL_GPIO_DeInit(GPIOB, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15);
00112
```

```
00113     HAL_DMA_DeInit(hspi->hdmatx);
00114     HAL_DMA_DeInit(hspi->hdmarx);
00115
00116     HAL_NVIC_DisableIRQ(SPI2_IRQn);
00117   }
00118 }
00119
00126 void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* htim_base) {
00127   if(htim_base->Instance==TIM2) {
00128     __HAL_RCC_TIM2_CLK_ENABLE();
00129     HAL_NVIC_SetPriority(TIM2_IRQn, 0, 0);
00130     HAL_NVIC_EnableIRQ(TIM2_IRQn);
00131   } else if(htim_base->Instance==TIM3) {
00132     __HAL_RCC_TIM3_CLK_ENABLE();
00133     HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);
00134     HAL_NVIC_EnableIRQ(TIM3_IRQn);
00135   }
00136 }
00137
00144 void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* htim_base) {
00145   if(htim_base->Instance==TIM2) {
00146     __HAL_RCC_TIM2_CLK_DISABLE();
00147     HAL_NVIC_DisableIRQ(TIM2_IRQn);
00148   } else if(htim_base->Instance==TIM3) {
00149     __HAL_RCC_TIM3_CLK_DISABLE();
00150     HAL_NVIC_DisableIRQ(TIM3_IRQn);
00151   }
00152 }
00153
00160 void HAL_UART_MspInit(UART_HandleTypeDef* huart) {
00161   GPIO_InitTypeDef GPIO_InitStruct = {0};
00162   if(huart->Instance==USART1) {
00163     __HAL_RCC_USART1_CLK_ENABLE();
00164     __HAL_RCC_GPIOB_CLK_ENABLE();
00165
00166     GPIO_InitStruct.Pin = GPIO_PIN_6;
00167     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00168     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
00169     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00170
00171     GPIO_InitStruct.Pin = GPIO_PIN_7;
00172     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00173     GPIO_InitStruct.Pull = GPIO_NOPULL;
00174     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00175
00176     __HAL_AFIO_REMAP_USART1_ENABLE();
00177   }
00178
00179 }
00180
00187 void HAL_UART_MspDeInit(UART_HandleTypeDef* huart) {
00188   if(huart->Instance==USART1) {
00189     __HAL_RCC_USART1_CLK_DISABLE();
00190
00191     HAL_GPIO_DeInit(GPIOB, GPIO_PIN_6|GPIO_PIN_7);
00192   }
00193
00194 }
```

9.27. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src/stm32f1xx_it.c

Interrupt Service Routines.

```
#include "main.h"
#include "stm32f1xx_it.h"
#include "../Modules/Gestor_SVM/GestorSVM.h"
```

Gráfico de dependencias incluidas en stm32f1xx_it.c:

Funciones

- void NMI_Handler (void)

This function handles Non maskable interrupt.

- void HardFault_Handler (void)

  This function handles Hard fault interrupt.
- void MemManage_Handler (void)

  This function handles Memory management fault.
- void BusFault_Handler (void)

  This function handles Prefetch fault, memory access fault.
- void UsageFault_Handler (void)

  This function handles Undefined instruction or illegal state.
- void SVC_Handler (void)

  This function handles System service call via SWI instruction.
- void DebugMon_Handler (void)

  This function handles Debug monitor.
- void PendSV_Handler (void)

  This function handles Pendable request for system service.
- void SysTick_Handler (void)

  This function handles System tick timer.
- void DMA1_Channel4_IRQHandler (void)

  This function handles DMA1 channel4 global interrupt.
- void DMA1_Channel5_IRQHandler (void)

  This function handles DMA1 channel5 global interrupt.
- void TIM2_IRQHandler (void)

  This function handles TIM2 global interrupt.
- void TIM3_IRQHandler (void)

  This function handles TIM3 global interrupt.
- void SPI2_IRQHandler (void)

  This function handles SPI2 global interrupt.

Variables

- DMA_HandleTypeDef hdma_spi2_tx
- DMA_HandleTypeDef hdma_spi2_rx
- SPI_HandleTypeDef hspi2
- TIM_HandleTypeDef htim2
- TIM_HandleTypeDef htim3

9.27.1. Descripción detallada

Interrupt Service Routines.

Atención

Definición en el archivo stm32f1xx_it.c.

9.27.2. Documentación de funciones

9.27.2.1. BusFault_Handler()

```
void BusFault_Handler (
             void )
```

This function handles Prefetch fault, memory access fault.

Definición en la línea 62 del archivo stm32f1xx_it.c.

9.27.2.2. DebugMon_Handler()

```
void DebugMon_Handler (
             void )
```

This function handles Debug monitor.

Definición en la línea 89 del archivo stm32f1xx_it.c.

9.27.2.3. DMA1_Channel4_IRQHandler()

```
void DMA1_Channel4_IRQHandler (
             void )
```

This function handles DMA1 channel4 global interrupt.

Definición en la línea 110 del archivo stm32f1xx_it.c.

9.27.2.4. DMA1_Channel5_IRQHandler()

```
void DMA1_Channel5_IRQHandler (
             void )
```

This function handles DMA1 channel5 global interrupt.

Definición en la línea 117 del archivo stm32f1xx_it.c.

9.27.2.5. HardFault_Handler()

```
void HardFault_Handler (
             void )
```

This function handles Hard fault interrupt.

Definición en la línea 42 del archivo stm32f1xx_it.c.

9.27.2.6. MemManage_Handler()

```
void MemManage_Handler (
            void )
```

This function handles Memory management fault.

Definición en la línea 52 del archivo stm32f1xx_it.c.

9.27.2.7. NMI_Handler()

```
void NMI_Handler (
            void )
```

This function handles Non maskable interrupt.

Definición en la línea 32 del archivo stm32f1xx_it.c.

9.27.2.8. PendSV_Handler()

```
void PendSV_Handler (
            void )
```

This function handles Pendable request for system service.

Definición en la línea 96 del archivo stm32f1xx_it.c.

9.27.2.9. SPI2_IRQHandler()

```
void SPI2_IRQHandler (
            void )
```

This function handles SPI2 global interrupt.

Definición en la línea 139 del archivo stm32f1xx_it.c.

9.27.2.10. SVC_Handler()

```
void SVC_Handler (
            void )
```

This function handles System service call via SWI instruction.

Definición en la línea 82 del archivo stm32f1xx_it.c.

9.27.2.11. SysTick_Handler()

```
void SysTick_Handler (
            void )
```

This function handles System tick timer.

Definición en la línea 103 del archivo stm32f1xx_it.c.

9.27.2.12. TIM2_IRQHandler()

```
void TIM2_IRQHandler (
            void )
```

This function handles TIM2 global interrupt.

Definición en la línea 124 del archivo stm32f1xx_it.c.

Gráfico de llamadas de esta función:

9.27.2.13. TIM3_IRQHandler()

```
void TIM3_IRQHandler (
            void )
```

This function handles TIM3 global interrupt.

Definición en la línea 132 del archivo stm32f1xx_it.c.

9.27.2.14. UsageFault_Handler()

```
void UsageFault_Handler (
            void )
```

This function handles Undefined instruction or illegal state.

Definición en la línea 72 del archivo stm32f1xx_it.c.

9.27.3. Documentación de variables

9.27.3.1. hdma_spi2_rx

```
DMA_HandleTypeDef hdma_spi2_rx  [extern]
```

Definición en la línea 44 del archivo main.c.

9.27.3.2. hdma_spi2_tx

```
DMA_HandleTypeDef hdma_spi2_tx  [extern]
```

Definición en la línea 43 del archivo main.c.

9.27.3.3. hspi2

```
SPI_HandleTypeDef hspi2  [extern]
```

Definición en la línea 42 del archivo main.c.

9.27.3.4. htim2

```
TIM_HandleTypeDef htim2  [extern]
```

Definición en la línea 45 del archivo main.c.

9.27.3.5. htim3

```
TIM_HandleTypeDef htim3  [extern]
```

Definición en la línea 46 del archivo main.c.

9.28. stm32f1xx_it.c

Ir a la documentación de este archivo.

```
00001 /* USER CODE BEGIN Header */
00017
00018 #include "main.h"
00019 #include "stm32f1xx_it.h"
00020
00021 #include "../Modules/Gestor_SVM/GestorSVM.h"
00022
00023 extern DMA_HandleTypeDef hdma_spi2_tx;
00024 extern DMA_HandleTypeDef hdma_spi2_rx;
00025 extern SPI_HandleTypeDef hspi2;
00026 extern TIM_HandleTypeDef htim2;
00027 extern TIM_HandleTypeDef htim3;
00028
00032 void NMI_Handler(void) {
00033
00034   while (1) {
00035
00036   }
00037 }
00038
00042 void HardFault_Handler(void) {
00043
00044   while (1) {
00045
00046   }
00047 }
00048
00052 void MemManage_Handler(void) {
00053
00054   while (1) {
00055
00056   }
00057 }
00058
00062 void BusFault_Handler(void) {
00063
00064   while (1) {
00065
00066   }
00067 }
00068
00072 void UsageFault_Handler(void) {
00073
00074   while (1) {
00075
00076   }
00077 }
00078
00082 void SVC_Handler(void) {
00083
00084 }
00085
00089 void DebugMon_Handler(void) {
00090
00091 }
00092
00096 void PendSV_Handler(void) {
```

```
00097
00098 }
00099
00103 void SysTick_Handler(void) {
00104   HAL_IncTick();
00105 }
00106
00110 void DMA1_Channel4_IRQHandler(void) {
00111   HAL_DMA_IRQHandler(&hdma_spi2_rx);
00112 }
00113
00117 void DMA1_Channel5_IRQHandler(void) {
00118   HAL_DMA_IRQHandler(&hdma_spi2_tx);
00119 }
00120
00124 void TIM2_IRQHandler(void) {
00125   GestorSVM_CalcInterrupt();
00126   HAL_TIM_IRQHandler(&htim2);
00127 }
00128
00132 void TIM3_IRQHandler(void) {
00133   HAL_TIM_IRQHandler(&htim3);
00134 }
00135
00139 void SPI2_IRQHandler(void) {
00140   HAL_SPI_IRQHandler(&hspi2);
00141 }
```

9.29. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src/syscalls.c

STM32CubeIDE Minimal System calls file.

```
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
```

Gráfico de dependencias incluidas en syscalls.c:

Funciones

- int __io_putchar (int ch) __attribute__((weak))
- int __io_getchar (void)
- void initialise_monitor_handles ()
- int _getpid (void)
- int _kill (int pid, int sig)
- void _exit (int status)
- __attribute__ ((weak))
- int _close (int file)
- int _fstat (int file, struct stat *st)
- int _isatty (int file)
- int _lseek (int file, int ptr, int dir)
- int _open (char *path, int flags,...)
- int _wait (int *status)
- int _unlink (char *name)
- int _times (struct tms *buf)
- int _stat (char *file, struct stat *st)
- int _link (char *old, char *new)
- int _fork (void)
- int _execve (char *name, char **argv, char **env)

Variables

- char ** environ = __env

9.29.1. Descripción detallada

STM32CubeIDE Minimal System calls file.

Autor

Auto-generated by STM32CubeIDE

```
For more information about which c-functions
need which of these lowlevel functions
please consult the Newlib libc-manual
```

Atención

Definición en el archivo syscalls.c.

9.29.2. Documentación de funciones

9.29.2.1. __attribute__()

```
__attribute__ (
             (weak) )
```

Definición en la línea 67 del archivo syscalls.c.

Gráfico de llamadas de esta función:

9.29.2.2. __io_getchar()

```
int __io_getchar (
             void )  [extern]
```

Definición en la línea 36 del archivo syscalls.c.

Gráfico de llamadas a esta función:

9.29.2.3. __io_putchar()

```
int __io_putchar (
             int ch)  [extern]
```

9.29.2.4. _close()

```
int _close (
            int file)
```

Definición en la línea 92 del archivo syscalls.c.

9.29.2.5. _execve()

```
int _execve (
            char * name,
            char ** argv,
            char ** env)
```

Definición en la línea 169 del archivo syscalls.c.

9.29.2.6. _exit()

```
void _exit (
            int status)
```

Definición en la línea 61 del archivo syscalls.c.

Gráfico de llamadas de esta función:

9.29.2.7. _fork()

```
int _fork (
            void )
```

Definición en la línea 163 del archivo syscalls.c.

9.29.2.8. _fstat()

```
int _fstat (
            int file,
            struct stat * st)
```

Definición en la línea 99 del archivo syscalls.c.

9.29.2.9. _getpid()

```
int _getpid (
            void )
```

Definición en la línea 48 del archivo syscalls.c.

9.29.2.10. _isatty()

```
int _isatty (
             int file)
```

Definición en la línea 106 del archivo syscalls.c.

9.29.2.11. _kill()

```
int _kill (
             int pid,
             int sig)
```

Definición en la línea 53 del archivo syscalls.c.

Gráfico de llamadas a esta función:

9.29.2.12. _link()

```
int _link (
             char * old,
             char * new)
```

Definición en la línea 155 del archivo syscalls.c.

9.29.2.13. _lseek()

```
int _lseek (
             int file,
             int ptr,
             int dir)
```

Definición en la línea 112 del archivo syscalls.c.

9.29.2.14. _open()

```
int _open (
             char * path,
             int flags,
              ...)
```

Definición en la línea 120 del archivo syscalls.c.

9.29.2.15. _stat()

```
int _stat (
             char * file,
             struct stat * st)
```

Definición en la línea 148 del archivo syscalls.c.

9.29.2.16. _times()

```
int _times (
              struct tms * buf)
```

Definición en la línea 142 del archivo syscalls.c.

9.29.2.17. _unlink()

```
int _unlink (
              char * name)
```

Definición en la línea 135 del archivo syscalls.c.

9.29.2.18. _wait()

```
int _wait (
              int * status)
```

Definición en la línea 128 del archivo syscalls.c.

9.29.2.19. initialise_monitor_handles()

```
void initialise_monitor_handles ()
```

Definición en la línea 44 del archivo syscalls.c.

9.29.3. Documentación de variables

9.29.3.1. environ

```
char** environ = __env
```

Definición en la línea 40 del archivo syscalls.c.

9.30. syscalls.c

Ir a la documentación de este archivo.

```
00001
00022
00023 /* Includes */
00024 #include <sys/stat.h>
00025 #include <stdlib.h>
00026 #include <errno.h>
00027 #include <stdio.h>
00028 #include <signal.h>
00029 #include <time.h>
00030 #include <sys/time.h>
00031 #include <sys/times.h>
00032
00033
00034 /* Variables */
00035 extern int __io_putchar(int ch) __attribute__((weak));
00036 extern int __io_getchar(void) __attribute__((weak));
00037
00038
00039 char *__env[1] = { 0 };
00040 char **environ = __env;
00041
00042
00043 /* Functions */
00044 void initialise_monitor_handles()
00045 {
00046 }
00047
00048 int _getpid(void)
00049 {
00050   return 1;
00051 }
00052
00053 int _kill(int pid, int sig)
00054 {
00055   (void)pid;
00056   (void)sig;
00057   errno = EINVAL;
00058   return -1;
00059 }
00060
00061 void _exit (int status)
00062 {
00063   _kill(status, -1);
00064   while (1) {}    /* Make sure we hang here */
00065 }
00066
00067 __attribute__((weak)) int _read(int file, char *ptr, int len)
00068 {
00069   (void)file;
00070   int DataIdx;
00071
00072   for (DataIdx = 0; DataIdx < len; DataIdx++)
00073   {
00074     *ptr++ = __io_getchar();
00075   }
00076
00077   return len;
00078 }
00079
00080 __attribute__((weak)) int _write(int file, char *ptr, int len)
00081 {
00082   (void)file;
00083   int DataIdx;
00084
00085   for (DataIdx = 0; DataIdx < len; DataIdx++)
00086   {
00087     __io_putchar(*ptr++);
00088   }
00089   return len;
00090 }
00091
00092 int _close(int file)
00093 {
00094   (void)file;
00095   return -1;
00096 }
00097
00098
00099 int _fstat(int file, struct stat *st)
00100 {
00101   (void)file;
00102   st->st_mode = S_IFCHR;
```

```
00103   return 0;
00104 }
00105
00106 int _isatty(int file)
00107 {
00108   (void)file;
00109   return 1;
00110 }
00111
00112 int _lseek(int file, int ptr, int dir)
00113 {
00114   (void)file;
00115   (void)ptr;
00116   (void)dir;
00117   return 0;
00118 }
00119
00120 int _open(char *path, int flags, ...)
00121 {
00122   (void)path;
00123   (void)flags;
00124   /* Pretend like we always fail */
00125   return -1;
00126 }
00127
00128 int _wait(int *status)
00129 {
00130   (void)status;
00131   errno = ECHILD;
00132   return -1;
00133 }
00134
00135 int _unlink(char *name)
00136 {
00137   (void)name;
00138   errno = ENOENT;
00139   return -1;
00140 }
00141
00142 int _times(struct tms *buf)
00143 {
00144   (void)buf;
00145   return -1;
00146 }
00147
00148 int _stat(char *file, struct stat *st)
00149 {
00150   (void)file;
00151   st->st_mode = S_IFCHR;
00152   return 0;
00153 }
00154
00155 int _link(char *old, char *new)
00156 {
00157   (void)old;
00158   (void)new;
00159   errno = EMLINK;
00160   return -1;
00161 }
00162
00163 int _fork(void)
00164 {
00165   errno = EAGAIN;
00166   return -1;
00167 }
00168
00169 int _execve(char *name, char **argv, char **env)
00170 {
00171   (void)name;
00172   (void)argv;
00173   (void)env;
00174   errno = ENOMEM;
00175   return -1;
00176 }
```

9.31. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src/sysmem.c

STM32CubeIDE System Memory calls file.

```
#include <errno.h>
#include <stdint.h>
```

Gráfico de dependencias incluidas en sysmem.c:

Funciones

- void * _sbrk (ptrdiff_t incr)

  _sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

Variables

- static uint8_t * __sbrk_heap_end = NULL

9.31.1. Descripción detallada

STM32CubeIDE System Memory calls file.

Autor

Generated by STM32CubeIDE

```
For more information about which C functions
need which of these lowlevel functions
please consult the newlib libc manual
```

Atención

Definición en el archivo sysmem.c.

9.31.2. Documentación de funciones

9.31.2.1. _sbrk()

```
void * _sbrk (
             ptrdiff_t incr)
```

_sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

```
* ##########################################################################
* #  .data  #  .bss  #       newlib heap       #          MSP stack          #
* #         #        #                         # Reserved by _Min_Stack_Size #
* ##########################################################################
* ^-- RAM start      ^-- _end                             _estack, RAM end --^
*
```

This implementation starts allocating at the '_end' linker symbol The '_Min_Stack_Size' linker symbol reserves a memory for the MSP stack The implementation considers '_estack' linker symbol to be RAM end NOTE: If the MSP stack, at any point during execution, grows larger than the reserved size, please increase the '_Min_Stack_Size'.

Parámetros

incr	Memory size

Devuelve

Pointer to allocated memory

Definición en la línea 53 del archivo sysmem.c.

9.31.3. Documentación de variables

9.31.3.1. __sbrk_heap_end

```
uint8_t* __sbrk_heap_end = NULL  [static]
```

Pointer to the current high watermark of the heap usage

Definición en la línea 30 del archivo sysmem.c.

9.32. sysmem.c

Ir a la documentación de este archivo.

```
00001
00022
00023 /* Includes */
00024 #include <errno.h>
00025 #include <stdint.h>
00026
00030 static uint8_t *__sbrk_heap_end = NULL;
00031
00053 void *_sbrk(ptrdiff_t incr)
00054 {
00055   extern uint8_t _end; /* Symbol defined in the linker script */
00056   extern uint8_t _estack; /* Symbol defined in the linker script */
00057   extern uint32_t _Min_Stack_Size; /* Symbol defined in the linker script */
00058   const uint32_t stack_limit = (uint32_t)&_estack - (uint32_t)&_Min_Stack_Size;
00059   const uint8_t *max_heap = (uint8_t *)stack_limit;
00060   uint8_t *prev_heap_end;
00061
00062   /* Initialize heap end at first call */
00063   if (NULL == __sbrk_heap_end)
00064   {
00065     __sbrk_heap_end = &_end;
00066   }
00067
00068   /* Protect heap from growing into the reserved MSP stack */
00069   if (__sbrk_heap_end + incr > max_heap)
00070   {
00071     errno = ENOMEM;
00072     return (void *)-1;
00073   }
00074
00075   prev_heap_end = __sbrk_heap_end;
00076   __sbrk_heap_end += incr;
00077
00078   return (void *)prev_heap_end;
00079 }
```

9.33. Referencia del archivo C:/Users/User/Desktop/ProyectoFinal-Grupo5-VariadorDeFrecuencia/Software/Firmware stm32/Src/system_stm32f1xx.c

CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.

```
#include "stm32f1xx.h"
```

Gráfico de dependencias incluidas en system_stm32f1xx.c:

defines

- #define HSE_VALUE 8000000U
- #define HSI_VALUE 8000000U

Funciones

- void SystemInit (void)

  Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Variables

- uint32_t SystemCoreClock = 8000000
- const uint8_t AHBPrescTable [16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t APBPrescTable [8U] = {0, 0, 0, 0, 1, 2, 3, 4}

9.33.1. Descripción detallada

CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.

Autor

MCD Application Team

1. This file provides two functions and one global variable to be called from user application:
   - SystemInit(): Setups the system clock (System clock source, PLL Multiplier factors, AHB/APBx prescalers and Flash settings). This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f1xx_xx.s" file.
   - SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
   - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.
2. After each device reset the HSI (8 MHz) is used as system clock source. Then SystemInit() function is called, in "startup_stm32f1xx_xx.s" file, to configure the system clock before to branch to main program.
3. The default value of HSE crystal is set to 8 MHz (or 25 MHz, depending on the product used), refer to "HSE_VALUE". When HSE is used as system clock source, directly or through PLL, and you are using different crystal you have to adapt the HSE value to your own configuration.

Atención

Definición en el archivo system_stm32f1xx.c.

9.34. system_stm32f1xx.c

Ir a la documentación de este archivo.

```
00001
00045
00049
00053
00057
00058 #include "stm32f1xx.h"
00059
00063
00067
00071
00075
00076 #if !defined  (HSE_VALUE)
00077   #define HSE_VALUE              8000000U
00079 #endif /* HSE_VALUE */
00080
00081 #if !defined  (HSI_VALUE)
00082   #define HSI_VALUE              8000000U
00084 #endif /* HSI_VALUE */
00085
00087 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
      defined(STM32F103xG)
00088 /* #define DATA_IN_ExtSRAM */
00089 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */
00090
00091 /* Note: Following vector table addresses must be defined in line with linker
00092          configuration. */
00096 /* #define USER_VECT_TAB_ADDRESS */
00097
00098 #if defined(USER_VECT_TAB_ADDRESS)
00101 /* #define VECT_TAB_SRAM */
00102 #if defined(VECT_TAB_SRAM)
00103 #define VECT_TAB_BASE_ADDRESS   SRAM_BASE
00105 #define VECT_TAB_OFFSET         0x00000000U
00107 #else
00108 #define VECT_TAB_BASE_ADDRESS   FLASH_BASE
00110 #define VECT_TAB_OFFSET         0x00000000U
00112 #endif /* VECT_TAB_SRAM */
00113 #endif /* USER_VECT_TAB_ADDRESS */
00114
00115 /******************************************************************************/
00116
00120
00124
00128
00132
00133   /* This variable is updated in three ways:
00134       1) by calling CMSIS function SystemCoreClockUpdate()
00135       2) by calling HAL API function HAL_RCC_GetHCLKFreq()
00136       3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency
00137          Note: If you use this function to configure the system clock; then there
00138                is no need to call the 2 first functions listed above, since SystemCoreClock
00139                variable is updated automatically.
00140   */
00141 uint32_t SystemCoreClock = 8000000;
00142 const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};
00143 const uint8_t APBPrescTable[8U] =  {0, 0, 0, 0, 1, 2, 3, 4};
00144
00148
00152
00153 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
      defined(STM32F103xG)
00154 #ifdef DATA_IN_ExtSRAM
```

```
00155   static void SystemInit_ExtMemCtl(void);
00156 #endif /* DATA_IN_ExtSRAM */
00157 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */
00158
00162
00166
00175 void SystemInit (void)
00176 {
00177 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
      defined(STM32F103xG)
00178   #ifdef DATA_IN_ExtSRAM
00179     SystemInit_ExtMemCtl();
00180   #endif /* DATA_IN_ExtSRAM */
00181 #endif
00182
00183   /* Configure the Vector Table location ----------------------------------*/
00184 #if defined(USER_VECT_TAB_ADDRESS)
00185   SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM. */
00186 #endif /* USER_VECT_TAB_ADDRESS */
00187 }
00188
00224 void SystemCoreClockUpdate (void)
00225 {
00226   uint32_t tmp = 0U, pllmull = 0U, pllsource = 0U;
00227
00228 #if defined(STM32F105xC) || defined(STM32F107xC)
00229   uint32_t prediv1source = 0U, prediv1factor = 0U, prediv2factor = 0U, pll2mull = 0U;
00230 #endif /* STM32F105xC */
00231
00232 #if defined(STM32F100xB) || defined(STM32F100xE)
00233   uint32_t prediv1factor = 0U;
00234 #endif /* STM32F100xB or STM32F100xE */
00235
00236   /* Get SYSCLK source -------------------------------------------------------*/
00237   tmp = RCC->CFGR & RCC_CFGR_SWS;
00238
00239   switch (tmp)
00240   {
00241     case 0x00U:  /* HSI used as system clock */
00242       SystemCoreClock = HSI_VALUE;
00243       break;
00244     case 0x04U:  /* HSE used as system clock */
00245       SystemCoreClock = HSE_VALUE;
00246       break;
00247     case 0x08U:  /* PLL used as system clock */
00248
00249       /* Get PLL clock source and multiplication factor ----------------------*/
00250       pllmull = RCC->CFGR & RCC_CFGR_PLLMULL;
00251       pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;
00252
00253 #if !defined(STM32F105xC) && !defined(STM32F107xC)
00254       pllmull = ( pllmull >> 18U) + 2U;
00255
00256       if (pllsource == 0x00U)
00257       {
00258         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00259         SystemCoreClock = (HSI_VALUE >> 1U) * pllmull;
00260       }
00261       else
00262       {
00263  #if defined(STM32F100xB) || defined(STM32F100xE)
00264        prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00265        /* HSE oscillator clock selected as PREDIV1 clock entry */
00266        SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00267  #else
00268        /* HSE selected as PLL clock entry */
00269        if ((RCC->CFGR & RCC_CFGR_PLLXTPRE) != (uint32_t)RESET)
00270        {/* HSE oscillator clock divided by 2 */
00271          SystemCoreClock = (HSE_VALUE >> 1U) * pllmull;
00272        }
00273        else
00274        {
00275          SystemCoreClock = HSE_VALUE * pllmull;
00276        }
00277  #endif
00278       }
00279 #else
00280       pllmull = pllmull >> 18U;
00281
00282       if (pllmull != 0x0DU)
00283       {
00284          pllmull += 2U;
00285       }
00286       else
00287       { /* PLL multiplication factor = PLL input clock * 6.5 */
00288         pllmull = 13U / 2U;
00289       }
```

```
00290
00291       if (pllsource == 0x00U)
00292       {
00293         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00294         SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
00295       }
00296       else
00297       {/* PREDIV1 selected as PLL clock entry */
00298
00299         /* Get PREDIV1 clock source and division factor */
00300         prediv1source = RCC->CFGR2 & RCC_CFGR2_PREDIV1SRC;
00301         prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00302
00303         if (prediv1source == 0U)
00304         {
00305           /* HSE oscillator clock selected as PREDIV1 clock entry */
00306           SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00307         }
00308         else
00309         {/* PLL2 clock selected as PREDIV1 clock entry */
00310
00311           /* Get PREDIV2 division factor and PLL2 multiplication factor */
00312           prediv2factor = ((RCC->CFGR2 & RCC_CFGR2_PREDIV2) » 4U) + 1U;
00313           pll2mull = ((RCC->CFGR2 & RCC_CFGR2_PLL2MUL) » 8U) + 2U;
00314           SystemCoreClock = (((HSE_VALUE / prediv2factor) * pll2mull) / prediv1factor) * pllmull;

00315         }
00316       }
00317 #endif /* STM32F105xC */
00318       break;
00319
00320     default:
00321       SystemCoreClock = HSI_VALUE;
00322       break;
00323   }
00324
00325   /* Compute HCLK clock frequency ----------------*/
00326   /* Get HCLK prescaler */
00327   tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) » 4U)];
00328   /* HCLK clock frequency */
00329   SystemCoreClock »= tmp;
00330 }
00331
00332 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
      defined(STM32F103xG)
00339 #ifdef DATA_IN_ExtSRAM
00349 void SystemInit_ExtMemCtl(void)
00350 {
00351   __IO uint32_t tmpreg;
00354
00355   /* Enable FSMC clock */
00356   RCC->AHBENR = 0x00000114U;
00357
00358   /* Delay after an RCC peripheral clock enabling */
00359   tmpreg = READ_BIT(RCC->AHBENR, RCC_AHBENR_FSMCEN);
00360
00361   /* Enable GPIOD, GPIOE, GPIOF and GPIOG clocks */
00362   RCC->APB2ENR = 0x000001E0U;
00363
00364   /* Delay after an RCC peripheral clock enabling */
00365   tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_IOPDEN);
00366
00367   (void)(tmpreg);
00368
00369 /* ---------------  SRAM Data lines, NOE and NWE configuration ---------------*/
00370 /*----------------  SRAM Address lines configuration -------------------------*/
00371 /*----------------  NOE and NWE configuration ---------------------------------*/
00372 /*----------------  NE3 configuration ------------------------------------------*/
00373 /*----------------  NBL0, NBL1 configuration ----------------------------------*/
00374
00375   GPIOD->CRL = 0x44BB44BBU;
00376   GPIOD->CRH = 0xBBBBBBBBU;
00377
00378   GPIOE->CRL = 0xB44444BBU;
00379   GPIOE->CRH = 0xBBBBBBBBU;
00380
00381   GPIOF->CRL = 0x44BBBBBBU;
00382   GPIOF->CRH = 0xBBBB4444U;
00383
00384   GPIOG->CRL = 0x44BBBBBBU;
00385   GPIOG->CRH = 0x444B4B44U;
00386
00387 /*----------------  FSMC Configuration ----------------------------------------*/
00388 /*----------------  Enable FSMC Bank1_SRAM Bank -------------------------------*/
00389
00390   FSMC_Bank1->BTCR[4U] = 0x00001091U;
00391   FSMC_Bank1->BTCR[5U] = 0x00110212U;
```

```
00392 }
00393 #endif /* DATA_IN_ExtSRAM */
00394 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */
00395
00399
00403
```