

Libreria LPC845

Generado por Doxygen 1.8.13



# Índice general

<b>1</b>	<b>Pagina principal de la documentacion</b>	<b>1</b>
1.1	Introduccion . . . . .	1
<b>2</b>	<b>Indice de módulos</b>	<b>3</b>
2.1	Módulos . . . . .	3
<b>3</b>	<b>Índice de estructura de datos</b>	<b>5</b>
3.1	Estructura de datos . . . . .	5
<b>4</b>	<b>Indice de archivos</b>	<b>7</b>
4.1	Lista de archivos . . . . .	7
<b>5</b>	<b>Documentación de módulos</b>	<b>9</b>
5.1	ADC . . . . .	9
5.1.1	Descripción detallada . . . . .	9
5.1.2	Documentación de las estructuras de datos . . . . .	11
5.1.2.1	struct hal_adc_sequence_result_t . . . . .	11
5.1.3	Documentación de las enumeraciones . . . . .	12
5.1.3.1	hal_adc_clock_source_en . . . . .	12
5.1.3.2	hal_adc_low_power_mode_en . . . . .	12
5.1.3.3	hal_adc_sequence_sel_en . . . . .	12
5.1.3.4	hal_adc_trigger_sel_en . . . . .	13
5.1.3.5	hal_adc_trigger_pol_sel_en . . . . .	13
5.1.3.6	hal_adc_sync_sel_en . . . . .	13
5.1.3.7	hal_adc_interrupt_mode_en . . . . .	14
5.1.3.8	hal_adc_result_channel_en . . . . .	14
5.1.3.9	hal_adc_sequence_result_en . . . . .	14
5.1.4	Documentación de las funciones . . . . .	15
5.1.4.1	hal_adc_init_async_mode() . . . . .	15
5.1.4.2	hal_adc_init_sync_mode() . . . . .	15
5.1.4.3	hal_adc_config_sequence() . . . . .	16
5.1.4.4	hal_adc_enable_sequence() . . . . .	16
5.1.4.5	hal_adc_start_sequence() . . . . .	17
5.1.4.6	hal_adc_get_sequence_result() . . . . .	17

<b>6 Documentación de las estructuras de datos</b>	<b>19</b>
6.1 Referencia de la Estructura <code>hal_adc_sequence_config_t</code>	19
6.1.1 Descripción detallada	19
6.1.2 Documentación de los campos	19
6.1.2.1 <code>channels</code>	20
6.1.2.2 <code>trigger</code>	20
6.1.2.3 <code>trigger_pol</code>	20
6.1.2.4 <code>sync_bypass</code>	20
6.1.2.5 <code>mode</code>	20
6.1.2.6 <code>single_step</code>	20
6.1.2.7 <code>low_priority</code>	21
6.1.2.8 <code>callback</code>	21
6.2 Referencia de la Estructura <code>hal_ctimer_match_config_t</code>	21
6.3 Referencia de la Estructura <code>hal_ctimer_pwm_channel_config_t</code>	21
6.4 Referencia de la Estructura <code>hal_ctimer_pwm_config_t</code>	22
6.5 Referencia de la Estructura <code>hal_pinint_config_t</code>	22
6.6 Referencia de la Estructura <code>hal_spi_master_mode_config_t</code>	22
6.7 Referencia de la Estructura <code>hal_uart_config_t</code>	23
<b>7 Documentación de archivos</b>	<b>25</b>
7.1 Referencia del Archivo <code>includes/hal/HAL_ADC.h</code>	25
7.1.1 Descripción detallada	26
7.2 Referencia del Archivo <code>includes/hal/HAL_TIMER.h</code>	27
7.2.1 Descripción detallada	28
7.2.2 Documentación de las funciones	28
7.2.2.1 <code>hal_ctimer_timer_mode_init()</code>	28
7.2.2.2 <code>hal_ctimer_timer_mode_config_match()</code>	29
7.2.2.3 <code>hal_ctimer_pwm_mode_init()</code>	29
7.2.2.4 <code>hal_ctimer_pwm_mode_set_period()</code>	29
7.2.2.5 <code>hal_ctimer_pwm_mode_config_channel()</code>	30
7.3 Referencia del Archivo <code>includes/hal/HAL_DAC.h</code>	30

7.3.1	Descripción detallada . . . . .	31
7.3.2	Documentación de las estructuras de datos . . . . .	31
7.3.2.1	struct hal_dac_ctrl_config_t . . . . .	31
7.3.3	Documentación de las funciones . . . . .	31
7.3.3.1	hal_dac_init() . . . . .	31
7.4	Referencia del Archivo includes/hal/HAL_GPIO.h . . . . .	32
7.4.1	Descripción detallada . . . . .	33
7.4.2	Documentación de las funciones . . . . .	33
7.4.2.1	hal_gpio_init() . . . . .	33
7.4.2.2	hal_gpio_set_dir() . . . . .	34
7.4.2.3	hal_gpio_set_pin() . . . . .	34
7.4.2.4	hal_gpio_clear_pin() . . . . .	34
7.4.2.5	hal_gpio_toggle_pin() . . . . .	35
7.4.2.6	hal_gpio_read_pin() . . . . .	35
7.5	Referencia del Archivo includes/hal/HAL_IOCON.h . . . . .	35
7.5.1	Descripción detallada . . . . .	36
7.5.2	Documentación de las estructuras de datos . . . . .	37
7.5.2.1	struct hal_iocon_config_t . . . . .	37
7.5.3	Documentación de las funciones . . . . .	37
7.5.3.1	hal_iocon_config_io() . . . . .	37
7.6	Referencia del Archivo includes/hal/HAL_PININT.h . . . . .	37
7.6.1	Descripción detallada . . . . .	39
7.6.2	Documentación de las funciones . . . . .	39
7.6.2.1	hal_pinint_configure_pin_interrupt() . . . . .	39
7.6.2.2	hal_pinint_register_callback() . . . . .	39
7.7	Referencia del Archivo includes/hal/HAL_SPI.h . . . . .	40
7.7.1	Descripción detallada . . . . .	41
7.7.2	Documentación de las estructuras de datos . . . . .	41
7.7.2.1	struct hal_spi_master_mode_tx_config_t . . . . .	41
7.7.2.2	struct hal_spi_master_mode_tx_data_t . . . . .	41

7.7.3	Documentación de las funciones . . . . .	42
7.7.3.1	hal_spi_master_mode_init() . . . . .	42
7.7.3.2	hal_spi_master_mode_rx_data() . . . . .	42
7.7.3.3	hal_spi_master_mode_config_tx() . . . . .	43
7.7.3.4	hal_spi_master_mode_tx_data() . . . . .	43
7.7.3.5	hal_spi_master_mode_register_tx_callback() . . . . .	43
7.7.3.6	hal_spi_master_mode_register_rx_callback() . . . . .	43
7.8	Referencia del Archivo includes/hal/HAL_SYSCON.h . . . . .	44
7.8.1	Descripción detallada . . . . .	46
7.8.2	Documentación de las funciones . . . . .	46
7.8.2.1	hal_syscon_get_system_clock() . . . . .	46
7.8.2.2	hal_syscon_get_fro_clock() . . . . .	46
7.8.2.3	hal_syscon_config_external_crystal() . . . . .	46
7.8.2.4	hal_syscon_config_fro_direct() . . . . .	47
7.8.2.5	hal_syscon_config_clkout() . . . . .	47
7.8.2.6	hal_syscon_config_frg() . . . . .	47
7.8.2.7	hal_syscon_set_peripheral_clock_source() . . . . .	49
7.8.2.8	hal_syscon_get_peripheral_clock() . . . . .	49
7.8.2.9	hal_syscon_set_iocon_glitch_divider() . . . . .	49
7.8.2.10	hal_syscon_config_pll() . . . . .	50
7.8.2.11	hal_syscon_get_pll_clock() . . . . .	50
7.9	Referencia del Archivo includes/hal/HAL_SYSTICK.h . . . . .	50
7.9.1	Descripción detallada . . . . .	51
7.9.2	Documentación de las funciones . . . . .	51
7.9.2.1	hal_systick_init() . . . . .	51
7.9.2.2	hal_systick_update_callback() . . . . .	52
7.10	Referencia del Archivo includes/hal/HAL_UART.h . . . . .	52
7.10.1	Descripción detallada . . . . .	53
7.10.2	Documentación de las funciones . . . . .	53
7.10.2.1	hal_uart_init() . . . . .	53
7.10.2.2	hal_uart_tx_byte() . . . . .	54
7.10.2.3	hal_uart_rx_byte() . . . . .	54
7.10.2.4	hal_uart_register_tx_callback() . . . . .	54
7.10.2.5	hal_uart_register_rx_callback() . . . . .	55
7.11	Referencia del Archivo includes/hal/HAL_WKT.h . . . . .	55
7.11.1	Descripción detallada . . . . .	56
7.11.2	Documentación de las funciones . . . . .	56
7.11.2.1	hal_wkt_init() . . . . .	56
7.11.2.2	hal_wkt_register_callback() . . . . .	57
<b>8</b>	<b>Documentación de ejemplos</b>	<b>59</b>
8.1	Ejemplo_ADC.c . . . . .	59

# Capítulo 1

## Pagina principal de la documentacion

### 1.1. Introduccion

Esta libreria esta pensada para... (bla bla bla)

Mañana mismo me pongo a escribir esto!





# Capítulo 2

## Índice de módulos

### 2.1. Módulos

Lista de todos los módulos:

ADC . . . . .	9
---------------	---



## Capítulo 3

# Índice de estructura de datos

### 3.1. Estructura de datos

Lista de estructuras con una breve descripción:

<a href="#">hal_adc_sequence_config_t</a>	19
<a href="#">hal_ctimer_match_config_t</a>	21
<a href="#">hal_ctimer_pwm_channel_config_t</a>	21
<a href="#">hal_ctimer_pwm_config_t</a>	22
<a href="#">hal_pinint_config_t</a>	22
<a href="#">hal_spi_master_mode_config_t</a>	22
<a href="#">hal_uart_config_t</a>	23



## Capítulo 4

# Indice de archivos

### 4.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

<a href="#">includes/hal/HAL_ADC.h</a>	
Declaraciones a nivel de aplicacion del periferico ADC (LPC845)	25
<a href="#">includes/hal/HAL_TIMER.h</a>	
Declaraciones a nivel de aplicacion del periferico CTIMER (LPC845)	27
<a href="#">includes/hal/HAL_DAC.h</a>	
Declaraciones a nivel de aplicacion del periferico DAC (LPC845)	30
<a href="#">includes/hal/HAL_GPIO.h</a>	
Declaraciones a nivel de aplicacion del periferico GPIO (LPC845)	32
<a href="#">includes/hal/HAL_IOCON.h</a>	
Declaraciones a nivel de aplicacion del periferico IOCON (LPC845)	35
<a href="#">includes/hal/HAL_PININT.h</a>	
Declaraciones a nivel de aplicacion del periferico PININT (LPC845)	37
<a href="#">includes/hal/HAL_SPI.h</a>	
Declaraciones a nivel de aplicacion del periferico SPI (LPC845)	40
<a href="#">includes/hal/HAL_SYSCON.h</a>	
Declaraciones a nivel de aplicacion del periferico SYSCON (LPC845)	44
<a href="#">includes/hal/HAL_SYSTICK.h</a>	
Declaraciones a nivel de aplicacion del periferico SYSICK (LPC845)	50
<a href="#">includes/hal/HAL_UART.h</a>	
Declaraciones a nivel de aplicacion del periferico UART (LPC845)	52
<a href="#">includes/hal/HAL_WKT.h</a>	
Declaraciones a nivel de aplicacion del periferico WKT (LPC845)	55



## Capítulo 5

# Documentación de módulos

### 5.1. ADC

#### 5.1.1. Descripción detallada

##### Descripción

Este periférico como su nombre lo indica, convierte una o más entradas analógicas, a un valor equivalente digital. En el caso del LPC845, tiene un único módulo *ADC* con una resolución de 12 bits, el cual tiene 12 canales, lo cual implica que se pueden realizar conversiones de 12 fuentes analógicas distintas, pero no así realizar conversiones *al mismo tiempo*. En caso de querer tomar señales de múltiples fuentes analógicas, se deberán hacer sucesivas conversiones en los distintos canales deseados.

Una resolución de 12 bits implica que la conversión aumentará cada unidad siguiendo la siguiente ecuación↔  
:  $ADC_{res} = \frac{V_{refp}}{2^N}$

Esto implica que podemos prever el valor resultante de la conversión analógica/digital mediante la siguiente ecuación:  $ADC_{conv} = \frac{V_{ADC_{in}}}{ADC_{res}}$

Cabe destacar, que las conversiones serán redondeadas **siempre** hacia abajo, es decir, se descartan los valores decimales.

##### Concepto de *Secuencia de conversión*

Para el *ADC* de este microcontrolador, un inicio de conversión en realidad puede implicar el inicio de una *secuencia de conversión*. Dicha secuencia puede implicar uno o más canales a convertir, y puede generar eventos tanto cuando se termina la secuencia completa, o cuando se termina cada canal de la secuencia. Asimismo los inicios de conversión pueden disparar una secuencia completa, o el próximo de los canales de dicha secuencia. Se tienen dos secuencias configurables (*Secuencia A* y *Secuencia B*), las cuales se pueden configurar de forma tal que una secuencia interrumpa a la otra.

##### Inicio de conversiones

El *ADC* de este microcontrolador permite el inicio de secuencia de conversión/canal de dos formas:

1. Iniciadas por software: Las secuencias de conversión son iniciadas mediante código.
2. Iniciadas por hardware: Las secuencias de conversión son iniciadas dependiendo de otras señales, sean las mismas internas o externas al microcontrolador.

## Calibración de hardware

Este periférico contiene un bloque de autocalibración, el cual debe ser utilizado luego de cada reinicio del microcontrolador o cada vez que se sale de modo de bajo consumo, para obtener la resolución y precisión especificada por el fabricante.

La librería implementa la calibración por hardware en la función `hal_adc_init`

## Velocidad de conversión

Cada conversión realizada toma un tiempo que dependerá del clock configurado en el periférico. Podemos obtener este tiempo de conversión mediante la ecuación:  $t_{conv\_ADC} = \frac{1}{25 * f_{ADC}}$

El multiplicador 25 en el denominador, es debido a la naturaleza del periférico de <e>aproximaciones sucesivas</e>. Esto implica que desde que se genera un inicio de conversión hasta que la misma finaliza, deben transcurrir 25 ciclos de clock del *ADC*.

Ejemplo: Configurando el *ADC* con una  $f_{ADC} = 25MHz$  obtenemos el tiempo tomado por cada conversión:

$$t_{conv\_ADC} = \frac{1}{25 * 1MHz}$$

$$t_{conv\_ADC} = 1\mu s$$

Esto implica que entre un inicio de conversión y la finalización de la misma, pasará  $1\mu s$ . Nótese que este tiempo corresponde a una conversión para un único canal. En caso de estar convirtiendo varios canales, se deberá multiplicar  $t_{conv\_ADC}$  por la cantidad de canales activos en la secuencia de conversión, para obtener el tiempo total desde un inicio de secuencia de conversión y la finalización de todos los canales.

## Ver también

Ejemplo\_ADC.c

## Estructuras de datos

- struct `hal_adc_sequence_config_t`
- struct `hal_adc_sequence_result_t`

## Enumeraciones

- enum `hal_adc_clock_source_en` { `HAL_ADC_CLOCK_SOURCE_FRO` = 0, `HAL_ADC_CLOCK_SYS_PLL` }
- enum `hal_adc_low_power_mode_en` { `HAL_ADC_LOW_POWER_MODE_DISABLED` = 0, `HAL_ADC_LOW_POWER_MODE_ENABLED` }
- enum `hal_adc_sequence_sel_en` { `HAL_ADC_SEQUENCE_SEL_A` = 0, `HAL_ADC_SEQUENCE_SEL_B` }
- enum `hal_adc_trigger_sel_en` { `HAL_ADC_TRIGGER_SEL_NONE` = 0, `HAL_ADC_TRIGGER_SEL_PININT0_IRQ`, `HAL_ADC_TRIGGER_SEL_PININT1_IRQ`, `HAL_ADC_TRIGGER_SEL_SCT0_OUT3`, `HAL_ADC_TRIGGER_SEL_SCT0_OUT4`, `HAL_ADC_TRIGGER_SEL_T0_MAT3`, `HAL_ADC_TRIGGER_SEL_CMP0_OUT_ADC`, `HAL_ADC_TRIGGER_SEL_GPIO_INT_BMAT`, `HAL_ADC_TRIGGER_SEL_ARM_TXEV` }
- enum `hal_adc_trigger_pol_sel_en` { `HAL_ADC_TRIGGER_POL_SEL_NEGATIVE_EDGE` = 0, `HAL_ADC_TRIGGER_POL_SEL_POSITIVE_EDGE` }



- enum `hal_adc_sync_sel_en` { `HAL_ADC_SYNC_SEL_ENABLE_SYNC` = 0, `HAL_ADC_SYNC_SEL_BYPASS_SYNC` }
- enum `hal_adc_interrupt_mode_en` { `HAL_ADC_INTERRUPT_MODE_EOC` = 0, `HAL_ADC_INTERRUPT_MODE_EOS` }
- enum `hal_adc_result_channel_en` { `HAL_ADC_RESULT_CHANNEL_0` = 0, `HAL_ADC_RESULT_CHANNEL_1`, `HAL_ADC_RESULT_CHANNEL_2`, `HAL_ADC_RESULT_CHANNEL_3`, `HAL_ADC_RESULT_CHANNEL_4`, `HAL_ADC_RESULT_CHANNEL_5`, `HAL_ADC_RESULT_CHANNEL_6`, `HAL_ADC_RESULT_CHANNEL_7`, `HAL_ADC_RESULT_CHANNEL_8`, `HAL_ADC_RESULT_CHANNEL_9`, `HAL_ADC_RESULT_CHANNEL_10`, `HAL_ADC_RESULT_CHANNEL_11`, `HAL_ADC_RESULT_CHANNEL_GLOBAL` }
- enum `hal_adc_sequence_result_en` { `HAL_ADC_SEQUENCE_RESULT_VALID` = 0, `HAL_ADC_SEQUENCE_RESULT_INVALID` }

## Funciones

- void `hal_adc_init_async_mode` (uint32\_t sample\_freq, uint8\_t div, `hal_adc_clock_source_en` clock\_source, `hal_adc_low_power_mode_en` low\_power)  
*Inicializar el ADC en modo **asíncrono**.*
- void `hal_adc_init_sync_mode` (uint32\_t sample\_freq, `hal_adc_low_power_mode_en` low\_power)  
*Inicializar el ADC en modo **síncrono**.*
- void `hal_adc_deinit` (void)  
*De-inicialización del ADC.*
- void `hal_adc_config_sequence` (`hal_adc_sequence_sel_en` sequence, const `hal_adc_sequence_config_t` \*config)  
*Configurar una secuencia de conversión.*
- void `hal_adc_enable_sequence` (`hal_adc_sequence_sel_en` sequence)  
*Habilitar una secuencia.*
- void `hal_adc_start_sequence` (`hal_adc_sequence_sel_en` sequence)  
*Disparar conversiones en una secuencia.*
- `hal_adc_sequence_result_en` `hal_adc_get_sequence_result` (`hal_adc_sequence_sel_en` sequence, `hal_adc_sequence_result_t` \*result)  
*Obtener resultado de la secuencia.*

### 5.1.2. Documentación de las estructuras de datos

#### 5.1.2.1. struct `hal_adc_sequence_result_t`

Dato que representa el resultado de una conversión (sea de secuencia completa o de canal)

Ejemplos:

`Ejemplo_ADC.c`.

Campos de datos

<code>hal_adc_result_channel_en</code>	channel	Canal que generó el resultado
<code>uint16_t</code>	result	Valor de la conversión

### 5.1.3. Documentación de las enumeraciones

#### 5.1.3.1. hal\_adc\_clock\_source\_en

enum `hal_adc_clock_source_en`

Selección de fuente de clock para el *ADC*

Valores de enumeraciones

HAL_ADC_CLOCK_SOURCE_FRO	Free running oscillator como fuente de clock
HAL_ADC_CLOCK_SYS_PLL	Phase locked loop oscillator como fuente de clock

#### 5.1.3.2. hal\_adc\_low\_power\_mode\_en

enum `hal_adc_low_power_mode_en`

Selección de modo bajo consumo

Valores de enumeraciones

HAL_ADC_LOW_POWER_MODE_DISABLED	Modo bajo consumo inhabilitado
HAL_ADC_LOW_POWER_MODE_ENABLED	Modo bajo consumo habilitado

#### 5.1.3.3. hal\_adc\_sequence\_sel\_en

enum `hal_adc_sequence_sel_en`

Selección de secuencia de *ADC*

Valores de enumeraciones

HAL_ADC_SEQUENCE_SEL↔ _A	Secuencia A
HAL_ADC_SEQUENCE_SEL↔ _B	Secuencia B

5.1.3.4. `hal_adc_trigger_sel_en`

```
enum hal_adc_trigger_sel_en
```

Fuente de trigger para el *ADC*

Valores de enumeraciones

<code>HAL_ADC_TRIGGER_SEL_NONE</code>	Ninguna (trigger por software)
<code>HAL_ADC_TRIGGER_SEL_PININT0_IRQ</code>	Interrupción de PININT, canal 0
<code>HAL_ADC_TRIGGER_SEL_PININT1_IRQ</code>	Interrupción de PININT, canal 1
<code>HAL_ADC_TRIGGER_SEL_SCT0_OUT3</code>	Salida 3 del SCT
<code>HAL_ADC_TRIGGER_SEL_SCT0_OUT4</code>	Salida 4 del SCT
<code>HAL_ADC_TRIGGER_SEL_T0_MAT3</code>	Match 3 del CTIMER
<code>HAL_ADC_TRIGGER_SEL_CMP0_OUT_ADC</code>	Salida 0 del comparador analógico
<code>HAL_ADC_TRIGGER_SEL_GPIO_INT_BMAT</code>	Pattern match
<code>HAL_ADC_TRIGGER_SEL_ARM_TXEV</code>	Señal TXEV causada por una instrucción SEV

5.1.3.5. `hal_adc_trigger_pol_sel_en`

```
enum hal_adc_trigger_pol_sel_en
```

Selección de polaridad del trigger del *ADC*

Valores de enumeraciones

<code>HAL_ADC_TRIGGER_POL_SEL_NEGATIVE_EDGE</code>	Flanco negativo
<code>HAL_ADC_TRIGGER_POL_SEL_POSITIVE_EDGE</code>	Flanco positivo

5.1.3.6. `hal_adc_sync_sel_en`

```
enum hal_adc_sync_sel_en
```

Selección de sincronismo en secuencia del *ADC*

Valores de enumeraciones

<code>HAL_ADC_SYNC_SEL_ENABLE_SYNC</code>	Habilitación de sincronismo
<code>HAL_ADC_SYNC_SEL_BYPASS_SYNC</code>	Bypass el sincronismo

#### 5.1.3.7. hal\_adc\_interrupt\_mode\_en

enum `hal_adc_interrupt_mode_en`

Selección de modo de interrupción del *ADC*

Valores de enumeraciones

HAL_ADC_INTERRUPT_MODE_EOC	Modo de interrupción en fin de conversión
HAL_ADC_INTERRUPT_MODE_EOS	Modo de interrupción en fin de secuencia

#### 5.1.3.8. hal\_adc\_result\_channel\_en

enum `hal_adc_result_channel_en`

Canal que genero el resultado de *ADC*

Valores de enumeraciones

HAL_ADC_RESULT_CHANNEL_0	Canal 0
HAL_ADC_RESULT_CHANNEL_1	Canal 1
HAL_ADC_RESULT_CHANNEL_2	Canal 2
HAL_ADC_RESULT_CHANNEL_3	Canal 3
HAL_ADC_RESULT_CHANNEL_4	Canal 4
HAL_ADC_RESULT_CHANNEL_5	Canal 5
HAL_ADC_RESULT_CHANNEL_6	Canal 6
HAL_ADC_RESULT_CHANNEL_7	Canal 7
HAL_ADC_RESULT_CHANNEL_8	Canal 8
HAL_ADC_RESULT_CHANNEL_9	Canal 9
HAL_ADC_RESULT_CHANNEL_10	Canal 10
HAL_ADC_RESULT_CHANNEL_11	Canal 11
HAL_ADC_RESULT_CHANNEL_GLOBAL	Global

#### 5.1.3.9. hal\_adc\_sequence\_result\_en

enum `hal_adc_sequence_result_en`

Resultado de obtención de resultado de secuencia

Valores de enumeraciones

HAL_ADC_SEQUENCE_RESULT_VALID	Resultado válido
HAL_ADC_SEQUENCE_RESULT_INVALID	Resultado inválido

### 5.1.4. Documentación de las funciones

#### 5.1.4.1. `hal_adc_init_async_mode()`

```
void hal_adc_init_async_mode (
    uint32_t sample_freq,
    uint8_t div,
    hal_adc_clock_source_en clock_source,
    hal_adc_low_power_mode_en low_power )
```

Inicializar el *ADC* en modo **asincrónico**.

Realiza la calibración de hardware y fija la frecuencia de muestreo deseada. Nota: Solamente se debe realizar el llamado a una de las dos funciones de inicialización del *ADC*

Ver también

[hal\\_adc\\_clock\\_source\\_en](#)  
[hal\\_adc\\_low\\_power\\_mode\\_en](#)

#### Parámetros

in	<i>sample_freq</i>	Frecuencia de sampleo deseada
in	<i>div</i>	Divisor para la lógica del <i>ADC</i>
in	<i>clock_source</i>	Fuente de clock para el <i>ADC</i>
in	<i>low_power</i>	Selección de modo de bajo consumo

#### 5.1.4.2. `hal_adc_init_sync_mode()`

```
void hal_adc_init_sync_mode (
    uint32_t sample_freq,
    hal_adc_low_power_mode_en low_power )
```

Inicializar el *ADC* en modo **sincrónico**.

Realiza la calibración de hardware y fija la frecuencia de muestreo deseada.

Ver también

[hal\\_adc\\_clock\\_source\\_en](#)  
[hal\\_adc\\_operation\\_mode\\_en](#)  
[hal\\_adc\\_low\\_power\\_mode\\_en](#)

#### Parámetros

in	<i>sample_freq</i>	Frecuencia de sampleo deseada
in	<i>low_power</i>	Selección de modo de bajo consumo

Ejemplos:

[Ejemplo\\_ADC.c](#).

#### 5.1.4.3. `hal_adc_config_sequence()`

```
void hal_adc_config_sequence (
    hal_adc_sequence_sel_en sequence,
    const hal_adc_sequence_config_t * config )
```

Configurar una secuencia de conversión.

Esta función no habilita la secuencia, al menos que el parametro **burst** este activo

Ver también

[hal\\_adc\\_sequence\\_sel\\_en](#)  
[hal\\_adc\\_sequence\\_config\\_t](#)

Parámetros

in	<i>sequence</i>	Selección de secuencia a configurar
in	<i>config</i>	Configuración deseada para la secuencia

Ejemplos:

[Ejemplo\\_ADC.c](#).

#### 5.1.4.4. `hal_adc_enable_sequence()`

```
void hal_adc_enable_sequence (
    hal_adc_sequence_sel_en sequence )
```

Habilitar una secuencia.

Ver también

[hal\\_adc\\_sequence\\_sel\\_en](#)

Parámetros

in	<i>sequence</i>	Secuencia a habilitar
----	-----------------	-----------------------

5.1.4.5. `hal_adc_start_sequence()`

```
void hal_adc_start_sequence (
    hal_adc_sequence_sel_en sequence )
```

Disparar conversiones en una secuencia.

La configuración de la secuencia, en particular el parametro **single\_step**, influye en si esta funcion dispara una secuencia entera o un paso de la misma.

Ver también

[hal\\_adc\\_sequence\\_sel\\_en](#)

Parámetros

in	<i>sequence</i>	Secuencia a disparar
----	-----------------	----------------------

Ejemplos:

[Ejemplo\\_ADC.c](#).

5.1.4.6. `hal_adc_get_sequence_result()`

```
hal_adc_sequence_result_en hal_adc_get_sequence_result (
    hal_adc_sequence_sel_en sequence,
    hal_adc_sequence_result_t * result )
```

Obtener resultado de la secuencia.

El comportamiento de esta funcion depende de la configuración de la secuencia, en particular de la configuracion **MODE**. En caso de estar configurada para interrumpir al final de cada conversión, la función únicamente guardara el resultado de la conversión en el primer lugar del parametro `<e>result</e>`, caso contrario, guardara la cantidad de canales habilitados en la conversión en los distintos lugares del parametro `<e>result</e>`.

Ver también

[hal\\_adc\\_sequence\\_result\\_en](#)  
[hal\\_adc\\_sequence\\_sel\\_en](#)  
[hal\\_adc\\_sequence\\_result\\_t](#)

Parámetros

in	<i>sequence</i>	Secuencia de la cual obtener el resultado
out	<i>result</i>	Lugares donde guardar los resultados de la secuencia

Devuelve

Resultado de la función

Ejemplos:

[Ejemplo\\_ADC.c](#).



## Capítulo 6

# Documentación de las estructuras de datos

### 6.1. Referencia de la Estructura `hal_adc_sequence_config_t`

```
#include <HAL_ADC.h>
```

#### Campos de datos

- `uint16_t channels`
- `hal_adc_trigger_sel_en trigger`
- `hal_adc_trigger_pol_sel_en trigger_pol`
- `hal_adc_sync_sel_en sync_bypass`
- `hal_adc_interrupt_mode_en mode`
- `uint8_t burst`
- `uint8_t single_step`
- `uint8_t low_priority`
- `void(* callback )(void)`

#### 6.1.1. Descripción detallada

Configuración de secuencia de *ADC*

Ejemplos:

[Ejemplo\\_ADC.c](#).

#### 6.1.2. Documentación de los campos

#### 6.1.2.1. channels

```
uint16_t hal_adc_sequence_config_t::channels
```

Canales habilitados. Cada uno de los bits representa el canal

Ejemplos:

[Ejemplo\\_ADC.c](#).

#### 6.1.2.2. trigger

```
hal_adc_trigger_sel_en hal_adc_sequence_config_t::trigger
```

Configuración de fuente de trigger para la secuencia

#### 6.1.2.3. trigger\_pol

```
hal_adc_trigger_pol_sel_en hal_adc_sequence_config_t::trigger_pol
```

Configuración de flanco del trigger para la secuencia

#### 6.1.2.4. sync\_bypass

```
hal_adc_sync_sel_en hal_adc_sequence_config_t::sync_bypass
```

Configuración de sincronismo de la secuencia

#### 6.1.2.5. mode

```
hal_adc_interrupt_mode_en hal_adc_sequence_config_t::mode
```

Configuración de modo de interrupcion

#### 6.1.2.6. single\_step

```
uint8_t hal_adc_sequence_config_t::single_step
```

Configuración de modo BURST. En caso de ser 0 esta inhabilitado, cualquier otro valor lo habilita Configuración de funcionamiento del trigger. En caso de ser 0, un trigger dispara la conversión de toda la secuencia configurada, en caso de ser cualquier otro valor, un trigger dispara la conversión del siguiente canal habilitado en la secuencia

### 6.1.2.7. `low_priority`

```
uint8_t hal_adc_sequence_config_t::low_priority
```

Fijar baja prioridad de la secuencia. Únicamente aplica para la secuencia A. En caso de ser 0, la secuencia A tiene prioridad por sobre el B, cualquier otro valor, implica que la secuencia B tiene prioridad por sobre la A

### 6.1.2.8. `callback`

```
void(* hal_adc_sequence_config_t::callback) (void)
```

Callback a ejecutar en interrupción de secuencia. La misma se generará al final de la conversión de cada canal, o de toda la secuencia, dependiendo de la configuración global del ADC

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_ADC.h`

## 6.2. Referencia de la Estructura `hal_ctimer_match_config_t`

### Campos de datos

- `uint8_t interrupt_on_match`
- `uint8_t reset_on_match`
- `uint8_t stop_on_match`
- `uint8_t reload_on_match`
- `uint32_t match_value_useg`
- `hal_ctimer_match_action_en match_action`
- `uint8_t enable_external_pin`
- `hal_gpio_portpin_en match_pin`
- `void(* callback)(void)`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_TIMER.h`

## 6.3. Referencia de la Estructura `hal_ctimer_pwm_channel_config_t`

### Campos de datos

- `uint8_t interrupt_on_action`
- `uint32_t duty`  
*Duty en decimas de por ciento (1 equivale a 0.1 %)*
- `hal_gpio_portpin_en channel_pin`
- `void(* callback)(void)`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_TIMER.h`

## 6.4. Referencia de la Estructura `hal_ctimer_pwm_config_t`

### Campos de datos

- `uint32_t clock_div`  
*Corresponde al numero deseado a dividir menos 1.*
- `uint32_t pwm_period_useg`  
*Periodo del PWM en microsegundos.*
- `uint8_t interrupt_on_period`
- `void(* callback)(void)`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_CTIMER.h`

## 6.5. Referencia de la Estructura `hal_pinint_config_t`

### Campos de datos

- `hal_pinint_channel_en channel`
- `hal_pinint_interrupt_mode_en mode`
- `hal_pinint_level_int_en int_on_level`
- `uint8_t int_on_rising_edge`
- `uint8_t int_on_falling_edge`
- `hal_gpio_portpin_en portpin`
- `void(* callback)(void)`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_PININT.h`

## 6.6. Referencia de la Estructura `hal_spi_master_mode_config_t`

### Campos de datos

- `hal_syscon_peripheral_clock_sel_en clock_source`
- `uint8_t pre_delay`
- `uint8_t post_delay`
- `uint8_t frame_delay`
- `uint8_t transfer_delay`
- `hal_gpio_portpin_en sck_portpin`
- `hal_gpio_portpin_en miso_portpin`
- `hal_gpio_portpin_en mosi_portpin`
- `hal_gpio_portpin_en ssel_portpin [4]`
- `hal_spi_ssel_polarity_en ssel_polarity [4]`
- `void(* tx_free_callback)(void)`
- `void(* rx_ready_callback)(void)`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_SPI.h`

## 6.7. Referencia de la Estructura `hal_uart_config_t`

### Campos de datos

- `hal_uart_data_len` en **data\_length**
- `hal_uart_parity` en **parity**
- `hal_uart_stop` en **stop\_bits**
- `hal_uart_oversampling` en **oversampling**
- `hal_syscon_peripheral_clock_sel` en **clock\_selection**
- `uint32_t` **baudrate**
- `hal_gpio_portpin` en **tx\_portpin**
- `hal_gpio_portpin` en **rx\_portpin**
- `void(* rx_ready_callback)(void)`
- `void(* tx_ready_callback)(void)`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `includes/hal/HAL_UART.h`



## Capítulo 7

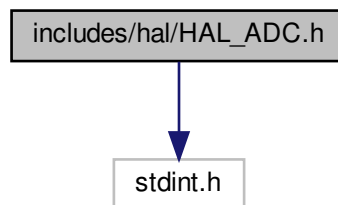
# Documentación de archivos

### 7.1. Referencia del Archivo includes/hal/HAL\_ADC.h

Declaraciones a nivel de aplicacion del periferico ADC (LPC845)

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

Dependencia gráfica adjunta para HAL\_ADC.h:



#### Estructuras de datos

- struct [hal\\_adc\\_sequence\\_config\\_t](#)
- struct [hal\\_adc\\_sequence\\_result\\_t](#)

#### Enumeraciones

- enum [hal\\_adc\\_clock\\_source\\_en](#) { [HAL\\_ADC\\_CLOCK\\_SOURCE\\_FRO](#) = 0, [HAL\\_ADC\\_CLOCK\\_SYS\\_PLL](#) }
- enum [hal\\_adc\\_low\\_power\\_mode\\_en](#) { [HAL\\_ADC\\_LOW\\_POWER\\_MODE\\_DISABLED](#) = 0, [HAL\\_ADC\\_LOW\\_POWER\\_MODE\\_ENABLED](#) }
- enum [hal\\_adc\\_sequence\\_sel\\_en](#) { [HAL\\_ADC\\_SEQUENCE\\_SEL\\_A](#) = 0, [HAL\\_ADC\\_SEQUENCE\\_SEL\\_B](#) }

- enum `hal_adc_trigger_sel_en` {  
`HAL_ADC_TRIGGER_SEL_NONE` = 0, `HAL_ADC_TRIGGER_SEL_PININT0_IRQ`, `HAL_ADC_TRIGGER_SEL_PININT1_IRQ`, `HAL_ADC_TRIGGER_SEL_SCT0_OUT3`,  
`HAL_ADC_TRIGGER_SEL_SCT0_OUT4`, `HAL_ADC_TRIGGER_SEL_T0_MAT3`, `HAL_ADC_TRIGGER_SEL_CMP0_OUT_ADC`, `HAL_ADC_TRIGGER_SEL_GPIO_INT_BMAT`,  
`HAL_ADC_TRIGGER_SEL_ARM_TXEV` }
- enum `hal_adc_trigger_pol_sel_en` { `HAL_ADC_TRIGGER_POL_SEL_NEGATIVE_EDGE` = 0, `HAL_ADC_TRIGGER_POL_SEL_POSITIVE_EDGE` }
- enum `hal_adc_sync_sel_en` { `HAL_ADC_SYNC_SEL_ENABLE_SYNC` = 0, `HAL_ADC_SYNC_SEL_BYPASS_SYNC` }
- enum `hal_adc_interrupt_mode_en` { `HAL_ADC_INTERRUPT_MODE_EOC` = 0, `HAL_ADC_INTERRUPT_MODE_EOS` }
- enum `hal_adc_result_channel_en` {  
`HAL_ADC_RESULT_CHANNEL_0` = 0, `HAL_ADC_RESULT_CHANNEL_1`, `HAL_ADC_RESULT_CHANNEL_2`, `HAL_ADC_RESULT_CHANNEL_3`,  
`HAL_ADC_RESULT_CHANNEL_4`, `HAL_ADC_RESULT_CHANNEL_5`, `HAL_ADC_RESULT_CHANNEL_6`, `HAL_ADC_RESULT_CHANNEL_7`,  
`HAL_ADC_RESULT_CHANNEL_8`, `HAL_ADC_RESULT_CHANNEL_9`, `HAL_ADC_RESULT_CHANNEL_10`, `HAL_ADC_RESULT_CHANNEL_11`,  
`HAL_ADC_RESULT_CHANNEL_GLOBAL` }
- enum `hal_adc_sequence_result_en` { `HAL_ADC_SEQUENCE_RESULT_VALID` = 0, `HAL_ADC_SEQUENCE_RESULT_INVALID` }

## Funciones

- void `hal_adc_init_async_mode` (uint32\_t sample\_freq, uint8\_t div, `hal_adc_clock_source_en` clock\_source, `hal_adc_low_power_mode_en` low\_power)  
*Inicializar el ADC en modo **asíncronico**.*
- void `hal_adc_init_sync_mode` (uint32\_t sample\_freq, `hal_adc_low_power_mode_en` low\_power)  
*Inicializar el ADC en modo **síncronico**.*
- void `hal_adc_deinit` (void)  
*De-inicialización del ADC.*
- void `hal_adc_config_sequence` (`hal_adc_sequence_sel_en` sequence, const `hal_adc_sequence_config_t` \*config)  
*Configurar una secuencia de conversión.*
- void `hal_adc_enable_sequence` (`hal_adc_sequence_sel_en` sequence)  
*Habilitar una secuencia.*
- void `hal_adc_start_sequence` (`hal_adc_sequence_sel_en` sequence)  
*Disparar conversiones en una secuencia.*
- `hal_adc_sequence_result_en` `hal_adc_get_sequence_result` (`hal_adc_sequence_sel_en` sequence, `hal_adc_sequence_result_t` \*result)  
*Obtener resultado de la secuencia.*

### 7.1.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico ADC (LPC845)

Autor

Augusto Santini



## Fecha

3/2020

## Versión

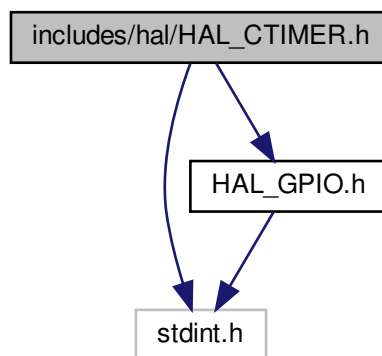
1.0

## 7.2. Referencia del Archivo includes/hal/HAL\_TIMER.h

Declaraciones a nivel de aplicación del periférico TIMER (LPC845)

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

Dependencia gráfica adjunta para HAL\_TIMER.h:



### Estructuras de datos

- struct `hal_timer_match_config_t`
- struct `hal_timer_pwm_channel_config_t`
- struct `hal_timer_pwm_config_t`

### Enumeraciones

- enum `hal_timer_match_action_en` { `HAL_TIMER_MATCH_DO_NOHING` = 0, `HAL_TIMER_MATCH_CLEAR_PIN`, `HAL_TIMER_MATCH_SET_PIN`, `HAL_TIMER_MATCH_TOGGLE_PIN` }
- enum `hal_timer_match_sel_en` { `HAL_TIMER_MATCH_0` = 0, `HAL_TIMER_MATCH_1`, `HAL_TIMER_MATCH_2`, `HAL_TIMER_MATCH_3` }
- enum `hal_timer_pwm_channel_sel_en` { `HAL_TIMER_PWM_CHANNEL_0` = 0, `HAL_TIMER_PWM_CHANNEL_1`, `HAL_TIMER_PWM_CHANNEL_2` }

## Funciones

- void `hal_ctimer_timer_mode_init` (uint32\_t clock\_div)  
*Inicializacion del periferico en modo timer.*
- void `hal_ctimer_timer_mode_config_match` (hal\_ctimer\_match\_sel\_en match\_sel, const `hal_ctimer_match_↵  
_config_t` \*match\_config)  
*Configurar un canal de match.*
- void `hal_ctimer_timer_mode_run` (void)  
*Habilitar el conteo del ctimer.*
- void `hal_ctimer_timer_mode_stop` (void)  
*Inhabilitar el conteo del ctimer.*
- void `hal_ctimer_timer_mode_reset` (void)  
*Reiniciar el conteo del ctimer.*
- void `hal_ctimer_pwm_mode_init` (const `hal_ctimer_pwm_config_t` \*config)  
*Inicializar el CTIMER en modo PWM.*
- void `hal_ctimer_pwm_mode_set_period` (uint32\_t period\_useg)  
*Actualizar el periodo en modo PWM.*
- void `hal_ctimer_pwm_mode_config_channel` (hal\_ctimer\_pwm\_channel\_sel\_en channel\_sel, const `hal_↵  
ctimer_pwm_channel_config_t` \*channel\_config)  
*Actualizar configuracion de algun canal de PWM.*

### 7.2.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico CTIMER (LPC845)

#### Autor

Augusto Santini

#### Fecha

3/2020

#### Versión

1.0

### 7.2.2. Documentación de las funciones

#### 7.2.2.1. `hal_ctimer_timer_mode_init()`

```
void hal_ctimer_timer_mode_init (
    uint32_t clock_div )
```

Inicializacion del periferico en modo timer.

Esta funcion no pone a correr el contador.

**Parámetros**

in	<i>clock_div</i>	Divisor del clock principal deseado (el valor efectivo es este valor + 1)
----	------------------	---

**7.2.2.2. hal\_ctimer\_timer\_mode\_config\_match()**

```
void hal_ctimer_timer_mode_config_match (
    hal_ctimer_match_sel_en match_sel,
    const hal_ctimer_match_config_t * match_config )
```

Configurar un canal de match.

**Parámetros**

in	<i>match_sel</i>	Match a configurar
in	<i>match_config</i>	Configuracion deseada

**7.2.2.3. hal\_ctimer\_pwm\_mode\_init()**

```
void hal_ctimer_pwm_mode_init (
    const hal_ctimer_pwm_config_t * config )
```

Inicializar el CTIMER en modo PWM.

**Parámetros**

in	<i>config</i>	Configuracion deseada
----	---------------	-----------------------

**7.2.2.4. hal\_ctimer\_pwm\_mode\_set\_period()**

```
void hal_ctimer_pwm_mode_set_period (
    uint32_t period_useg )
```

Actualizar el periodo en modo PWM.

**Parámetros**

in	<i>period_useg</i>	Nuevo periodo deseado en microsegundos
----	--------------------	--

### 7.2.2.5. hal\_ctimer\_pwm\_mode\_config\_channel()

```
void hal_ctimer_pwm_mode_config_channel (
    hal_ctimer_pwm_channel_sel_en channel_sel,
    const hal_ctimer_pwm_channel_config_t * channel_config )
```

Actualizar configuracion de algun canal de PWM.

#### Parámetros

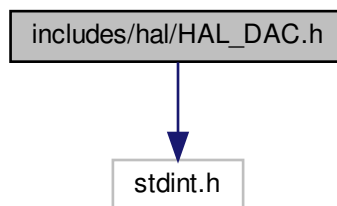
in	<i>channel_sel</i>	Seleccion de canal a configurar
in	<i>channel_config</i>	Configuracion del canal de PWM

## 7.3. Referencia del Archivo includes/hal/HAL\_DAC.h

Declaraciones a nivel de aplicacion del periferico DAC (LPC845)

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

Dependencia gráfica adjunta para HAL\_DAC.h:



### Estructuras de datos

- struct [hal\\_dac\\_ctrl\\_config\\_t](#)

### Enumeraciones

- enum **hal\_dac\_en** { HAL\_DAC\_0 = 0, HAL\_DAC\_1 }
- enum **hal\_dac\_settling\_time\_en** { HAL\_DAC\_SETTLING\_TIME\_1US\_MAX = 0, HAL\_DAC\_SETTLING\_TIME\_2\_5US\_MAX }

### Funciones

- void [hal\\_dac\\_init](#) (hal\_dac\_en dac, hal\_dac\_settling\_time\_en settling\_time, uint32\_t initial\_value)  
*Inicializacion del DAC.*

### 7.3.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico DAC (LPC845)

**Autor**

Augusto Santini

**Fecha**

3/2020

**Versión**

1.0

### 7.3.2. Documentación de las estructuras de datos

#### 7.3.2.1. struct hal\_dac\_ctrl\_config\_t

**Campos de datos**

uint8_t	count_enable: 1	
uint8_t	double_buffering: 1	
uint8_t	dma_enable: 1	
uint8_t	dma_request: 1	

### 7.3.3. Documentación de las funciones

#### 7.3.3.1. hal\_dac\_init()

```
void hal_dac_init (
    hal_dac_en dac,
    hal_dac_settling_time_en settling_time,
    uint32_t initial_value )
```

Inicializacion del DAC.

**Parámetros**

in	<i>dac</i>	Cual de los dos DACs inicializar
in	<i>settling_time</i>	Velocidad de conversion del DAC
in	<i>initial_value</i>	Valor inicial del DAC

## 7.4. Referencia del Archivo includes/hal/HAL\_GPIO.h

Declaraciones a nivel de aplicacion del periferico GPIO (LPC845)

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

Dependencia gráfica adjunta para HAL\_GPIO.h:

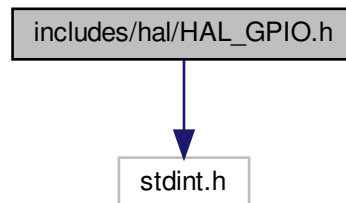
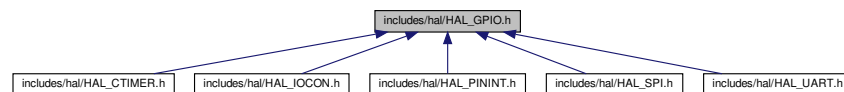


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



### defines

- #define HAL\_GPIO\_PORTPIN\_TO\_PORT(x) (x / 32)
- #define HAL\_GPIO\_PORTPIN\_TO\_PIN(x) (x % 32)

### Enumeraciones

- enum hal\_gpio\_port\_en { HAL\_GPIO\_PORT\_0 = 0, HAL\_GPIO\_PORT\_1 }
- enum hal\_gpio\_portpin\_en {
 HAL\_GPIO\_PORTPIN\_0\_0 = 0, HAL\_GPIO\_PORTPIN\_0\_1, HAL\_GPIO\_PORTPIN\_0\_2, HAL\_GPIO\_PORTPIN\_0\_3,
 HAL\_GPIO\_PORTPIN\_0\_4, HAL\_GPIO\_PORTPIN\_0\_5, HAL\_GPIO\_PORTPIN\_0\_6, HAL\_GPIO\_PORTPIN\_0\_7,
 HAL\_GPIO\_PORTPIN\_0\_8, HAL\_GPIO\_PORTPIN\_0\_9, HAL\_GPIO\_PORTPIN\_0\_10, HAL\_GPIO\_PORTPIN\_0\_11,
 HAL\_GPIO\_PORTPIN\_0\_12, HAL\_GPIO\_PORTPIN\_0\_13, HAL\_GPIO\_PORTPIN\_0\_14, HAL\_GPIO\_PORTPIN\_0\_15,
 HAL\_GPIO\_PORTPIN\_0\_16, HAL\_GPIO\_PORTPIN\_0\_17, HAL\_GPIO\_PORTPIN\_0\_18, HAL\_GPIO\_PORTPIN\_0\_19,
 HAL\_GPIO\_PORTPIN\_0\_20, HAL\_GPIO\_PORTPIN\_0\_21, HAL\_GPIO\_PORTPIN\_0\_22, HAL\_GPIO\_PORTPIN\_0\_23,
 }

- ```

HAL_GPIO_PORTPIN_0_24, HAL_GPIO_PORTPIN_0_25, HAL_GPIO_PORTPIN_0_26, HAL_GPIO_P↵
ORTPIN_0_27,
HAL_GPIO_PORTPIN_0_28, HAL_GPIO_PORTPIN_0_29, HAL_GPIO_PORTPIN_0_30, HAL_GPIO_P↵
ORTPIN_0_31,
HAL_GPIO_PORTPIN_1_0, HAL_GPIO_PORTPIN_1_1, HAL_GPIO_PORTPIN_1_2, HAL_GPIO_POR↵
TPIN_1_3,
HAL_GPIO_PORTPIN_1_4, HAL_GPIO_PORTPIN_1_5, HAL_GPIO_PORTPIN_1_6, HAL_GPIO_POR↵
TPIN_1_7,
HAL_GPIO_PORTPIN_1_8, HAL_GPIO_PORTPIN_1_9, HAL_GPIO_PORTPIN_NOT_USED }
▪ enum hal_gpio_dir_en { HAL_GPIO_DIR_INPUT = 0, HAL_GPIO_DIR_OUTPUT }

```

## Funciones

- void [hal\\_gpio\\_init](#) (hal\_gpio\_port\_en port)  
*Inicializar un puerto.*
- void [hal\\_gpio\\_set\\_dir](#) (hal\_gpio\_portpin\_en portpin, hal\_gpio\_dir\_en dir, uint8\_t initial\_state)  
*Fijar direccion de una GPIO.*
- void [hal\\_gpio\\_set\\_pin](#) (hal\_gpio\_portpin\_en portpin)  
*Fijar estado activo de una GPIO.*
- void [hal\\_gpio\\_clear\\_pin](#) (hal\_gpio\_portpin\_en portpin)  
*Fijar estado inactivo de una GPIO.*
- void [hal\\_gpio\\_toggle\\_pin](#) (hal\_gpio\_portpin\_en portpin)  
*Invertir estado de una GPIO.*
- uint8\_t [hal\\_gpio\\_read\\_pin](#) (hal\_gpio\_portpin\_en portpin)  
*Leer el estado de una GPIO.*

### 7.4.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico GPIO (LPC845)

#### Autor

Augusto Santini

#### Fecha

3/2020

#### Versión

1.0

### 7.4.2. Documentación de las funciones

#### 7.4.2.1. [hal\\_gpio\\_init\(\)](#)

```

void hal_gpio_init (
    hal_gpio_port_en port )

```

Inicializar un puerto.

**Parámetros**

|    |             |                      |
|----|-------------|----------------------|
| in | <i>port</i> | Puerto a inicializar |
|----|-------------|----------------------|

**7.4.2.2. hal\_gpio\_set\_dir()**

```
void hal_gpio_set_dir (
    hal_gpio_portpin_en portpin,
    hal_gpio_dir_en dir,
    uint8_t initial_state )
```

Fijar direccion de una GPIO.

**Parámetros**

|    |                      |                                               |
|----|----------------------|-----------------------------------------------|
| in | <i>portpin</i>       | Numero de puerto/pin a configurar             |
| in | <i>dir</i>           | Direccion deseada                             |
| in | <i>initial_state</i> | Estado inicial (aplica para salidas nada mas) |

**7.4.2.3. hal\_gpio\_set\_pin()**

```
void hal_gpio_set_pin (
    hal_gpio_portpin_en portpin )
```

Fijar estado activo de una GPIO.

**Parámetros**

|    |                |                                 |
|----|----------------|---------------------------------|
| in | <i>portpin</i> | Numero de puerto/pin a accionar |
|----|----------------|---------------------------------|

**7.4.2.4. hal\_gpio\_clear\_pin()**

```
void hal_gpio_clear_pin (
    hal_gpio_portpin_en portpin )
```

Fijar estado inactivo de una GPIO.

**Parámetros**

|    |                |                                 |
|----|----------------|---------------------------------|
| in | <i>portpin</i> | Numero de puerto/pin a accionar |
|----|----------------|---------------------------------|



#### 7.4.2.5. hal\_gpio\_toggle\_pin()

```
void hal_gpio_toggle_pin (
    hal_gpio_portpin_en portpin )
```

Invertir estado de una GPIO.

##### Parámetros

|    |                |                                 |
|----|----------------|---------------------------------|
| in | <i>portpin</i> | Numero de puerto/pin a accionar |
|----|----------------|---------------------------------|

#### 7.4.2.6. hal\_gpio\_read\_pin()

```
uint8_t hal_gpio_read_pin (
    hal_gpio_portpin_en portpin )
```

Leer el estado de una GPIO.

##### Parámetros

|    |                |                                 |
|----|----------------|---------------------------------|
| in | <i>portpin</i> | Numero de puerto/pin a accionar |
|----|----------------|---------------------------------|

##### Devuelve

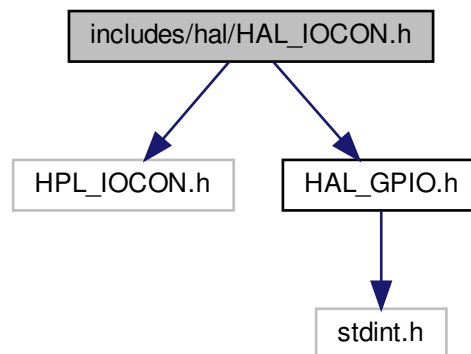
Estado actual de la GPIO

## 7.5. Referencia del Archivo includes/hal/HAL\_IOCON.h

Declaraciones a nivel de aplicacion del periferico IOCON (LPC845)

```
#include <HPL_IOCON.h>
#include <HAL_GPIO.h>
```

Dependencia gráfica adjunta para HAL\_IOCON.h:



## Estructuras de datos

- struct [hal\\_iocon\\_config\\_t](#)

## Enumeraciones

- enum [hal\\_iocon\\_pull\\_mode\\_en](#) { HAL\_IOCON\_PULL\_NONE = 0, HAL\_IOCON\_PULL\_DOWN, HAL\_IOCON\_PULL\_UP, HAL\_IOCON\_PULL\_REPEATER }
- enum [hal\\_iocon\\_sample\\_mode\\_en](#) { HAL\_IOCON\_SAMPLE\_MODE\_BYPASS = 0, HAL\_IOCON\_SAMPLE\_MODE\_1\_CLOCK, HAL\_IOCON\_SAMPLE\_MODE\_2\_CLOCK, HAL\_IOCON\_SAMPLE\_MODE\_3\_CLOCK }
- enum [hal\\_iocon\\_clk\\_sel\\_en](#) { HAL\_IOCON\_CLK\_DIV\_0 = 0, HAL\_IOCON\_CLK\_DIV\_1, HAL\_IOCON\_CLK\_DIV\_2, HAL\_IOCON\_CLK\_DIV\_3, HAL\_IOCON\_CLK\_DIV\_4, HAL\_IOCON\_CLK\_DIV\_5, HAL\_IOCON\_CLK\_DIV\_6 }
- enum [hal\\_iocon\\_iic\\_mode\\_en](#) { HAL\_IOCON\_IIC\_MODE\_STANDARD = 0, HAL\_IOCON\_IIC\_MODE\_GPIO, HAL\_IOCON\_IIC\_MODE\_FAST\_MODE }

## Funciones

- void [hal\\_iocon\\_config\\_io](#) (hal\_gpio\_portpin\_en portpin, const [hal\\_iocon\\_config\\_t](#) \*config)  
*Configuración de un pin.*

### 7.5.1. Descripción detallada

Declaraciones a nivel de aplicación del periférico IOCON (LPC845)

#### Autor

Augusto Santini

## Fecha

3/2020

## Versión

1.0

## 7.5.2. Documentación de las estructuras de datos

## 7.5.2.1. struct hal\_iocon\_config\_t

## Campos de datos

|                          |              |  |
|--------------------------|--------------|--|
| hal_iocon_pull_mode_en   | pull_mode    |  |
| uint8_t                  | hysteresis   |  |
| uint8_t                  | invert_input |  |
| uint8_t                  | open_drain   |  |
| hal_iocon_sample_mode_en | sample_mode  |  |
| hal_iocon_clk_sel_en     | clk_sel      |  |
| uint8_t                  | dac_mode     |  |
| hal_iocon_iic_mode_en    | iic_mode     |  |

## 7.5.3. Documentación de las funciones

## 7.5.3.1. hal\_iocon\_config\_io()

```
void hal_iocon_config_io (
    hal_gpio_portpin_en portpin,
    const hal_iocon_config_t * config )
```

Configuracion de un pin.

## Parámetros

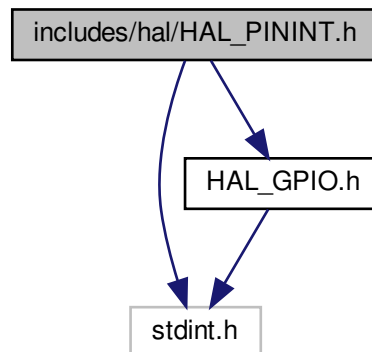
|    |                   |                                               |
|----|-------------------|-----------------------------------------------|
| in | <i>portpin</i>    | Puerto/pin a configurar                       |
| in | <i>pin_config</i> | Puntero a estructura de configuracion del pin |

## 7.6. Referencia del Archivo includes/hal/HAL\_PININT.h

Declaraciones a nivel de aplicacion del periferico PININT (LPC845)

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

Dependencia gráfica adjunta para HAL\_PININT.h:



## Estructuras de datos

- struct [hal\\_pinint\\_config\\_t](#)

## Enumeraciones

- enum [hal\\_pinint\\_channel\\_en](#) { [HAL\\_PININT\\_CHANNEL\\_0](#) = 0, [HAL\\_PININT\\_CHANNEL\\_1](#), [HAL\\_PININT\\_CHANNEL\\_2](#), [HAL\\_PININT\\_CHANNEL\\_3](#), [HAL\\_PININT\\_CHANNEL\\_4](#), [HAL\\_PININT\\_CHANNEL\\_5](#), [HAL\\_PININT\\_CHANNEL\\_6](#), [HAL\\_PININT\\_CHANNEL\\_7](#) }
- enum [hal\\_pinint\\_interrupt\\_mode\\_en](#) { [HAL\\_PININT\\_INTERRUPT\\_MODE\\_EDGE](#) = 0, [HAL\\_PININT\\_INTERRUPT\\_MODE\\_LEVEL](#) }
- enum [hal\\_pinint\\_level\\_int\\_en](#) { [HAL\\_PININT\\_LEVEL\\_INT\\_HIGH](#) = 0, [HAL\\_PININT\\_LEVEL\\_INT\\_LOW](#) }

## Funciones

- void [hal\\_pinint\\_init](#) (void)  
*Inicializacion del modulo.*
- void [hal\\_pinint\\_configure\\_pin\\_interrupt](#) (const [hal\\_pinint\\_config\\_t](#) \*config)  
*Configurar interrupciones de pin.*
- void [hal\\_pinint\\_register\\_callback](#) ([hal\\_pinint\\_channel\\_en](#) channel, void(\*new\_callback)(void))  
*Registrar callback a llamar en interrupcion de PININTn.*

### 7.6.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico PININT (LPC845)

**Autor**

Augusto Santini

**Fecha**

3/2020

**Versión**

1.0

### 7.6.2. Documentación de las funciones

#### 7.6.2.1. hal\_pinint\_configure\_pin\_interrupt()

```
void hal_pinint_configure_pin_interrupt (
    const hal_pinint_config_t * config )
```

Configurar interrupciones de pin.

**Parámetros**

|    |               |                                        |
|----|---------------|----------------------------------------|
| in | <i>config</i> | Configuracion de interrupciones de pin |
|----|---------------|----------------------------------------|

#### 7.6.2.2. hal\_pinint\_register\_callback()

```
void hal_pinint_register_callback (
    hal_pinint_channel_en channel,
    void(*) (void) new_callback )
```

Registrar callback a llamar en interrupcion de PININTn.

**Parámetros**

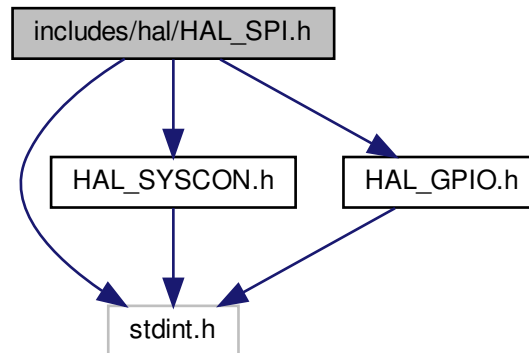
|    |                     |                                     |
|----|---------------------|-------------------------------------|
| in | <i>channel</i>      | Canal al cual registrar el callback |
| in | <i>new_callback</i> | Puntero a funcion a ejecutar        |

## 7.7. Referencia del Archivo includes/hal/HAL\_SPI.h

Declaraciones a nivel de aplicación del periférico SPI (LPC845)

```
#include <stdint.h>
#include <HAL_SYSCON.h>
#include <HAL_GPIO.h>
```

Dependencia gráfica adjunta para HAL\_SPI.h:



### Estructuras de datos

- struct `hal_spi_master_mode_config_t`
- struct `hal_spi_master_mode_tx_config_t`
- struct `hal_spi_master_mode_tx_data_t`

### defines

- `#define HAL_SPI_DUMMY_BYTE (0xFF)`

### Enumeraciones

- enum `hal_spi_sel_en` { `HAL_SPI_0` = 0, `HAL_SPI_1` }
- enum `hal_spi_data_length_en` { `HAL_SPI_DATA_LENGTH_1_BIT` = 0, `HAL_SPI_DATA_LENGTH_2_BIT`, `HAL_SPI_DATA_LENGTH_3_BIT`, `HAL_SPI_DATA_LENGTH_4_BIT`, `HAL_SPI_DATA_LENGTH_5_BIT`, `HAL_SPI_DATA_LENGTH_6_BIT`, `HAL_SPI_DATA_LENGTH_7_BIT`, `HAL_SPI_DATA_LENGTH_8_BIT`, `HAL_SPI_DATA_LENGTH_9_BIT`, `HAL_SPI_DATA_LENGTH_10_BIT`, `HAL_SPI_DATA_LENGTH_11_BIT`, `HAL_SPI_DATA_LENGTH_12_BIT`, `HAL_SPI_DATA_LENGTH_13_BIT`, `HAL_SPI_DATA_LENGTH_14_BIT`, `HAL_SPI_DATA_LENGTH_15_BIT`, `HAL_SPI_DATA_LENGTH_16_BIT` }
- enum `hal_spi_clock_mode_en` { `HAL_SPI_CLOCK_MODE_0` = 0, `HAL_SPI_CLOCK_MODE_1`, `HAL_SPI_CLOCK_MODE_2`, `HAL_SPI_CLOCK_MODE_3` }
- enum `hal_spi_ssel_polarity_en` { `HAL_SPI_SSEL_POLARITY_LOW` = 0, `HAL_SPI_SSEL_POLARITY_HIGH` }
- enum `hal_spi_ssel_sel_en` { `HAL_SPI_SSEL_SELECTION_0` = 0, `HAL_SPI_SSEL_SELECTION_1`, `HAL_SPI_SSEL_SELECTION_2`, `HAL_SPI_SSEL_SELECTION_3`, `HAL_SPI_SSEL_SELECTION_OTHER` }

## Funciones

- void [hal\\_spi\\_master\\_mode\\_init](#) (hal\_spi\_sel\_en inst, const [hal\\_spi\\_master\\_mode\\_config\\_t](#) \*config)  
*Inicializar SPI en modo master.*
- uint16\_t [hal\\_spi\\_master\\_mode\\_rx\\_data](#) (hal\_spi\_sel\_en inst)  
*Leer el dato recibido.*
- void [hal\\_spi\\_master\\_mode\\_config\\_tx](#) (hal\_spi\_sel\_en inst, const [hal\\_spi\\_master\\_mode\\_tx\\_config\\_t](#) \*config)  
*Configurar la transmision.*
- void [hal\\_spi\\_master\\_mode\\_tx\\_data](#) (hal\_spi\_sel\_en inst, const [hal\\_spi\\_master\\_mode\\_tx\\_data\\_t](#) \*data)  
*Transmitir dato.*
- void [hal\\_spi\\_master\\_mode\\_register\\_tx\\_callback](#) (hal\_spi\_sel\_en inst, void(\*new\_callback)(void))  
*Actualizar callback en TXRDY.*
- void [hal\\_spi\\_master\\_mode\\_register\\_rx\\_callback](#) (hal\_spi\_sel\_en inst, void(\*new\_callback)(void))  
*Actualizar callback en RXRDY.*

### 7.7.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico SPI (LPC845)

#### Autor

Augusto Santini

#### Fecha

3/2020

#### Versión

1.0

### 7.7.2. Documentación de las estructuras de datos

#### 7.7.2.1. struct [hal\\_spi\\_master\\_mode\\_tx\\_config\\_t](#)

##### Campos de datos

|                                       |            |  |
|---------------------------------------|------------|--|
| <a href="#">hal_spi_clock_mode_en</a> | clock_mode |  |
| uint16_t                              | clock_div  |  |

#### 7.7.2.2. struct [hal\\_spi\\_master\\_mode\\_tx\\_data\\_t](#)

##### Campos de datos

|          |            |  |
|----------|------------|--|
| uint32_t | data: 16   |  |
| uint32_t | ssel0_n: 1 |  |
| uint32_t | ssel1_n: 1 |  |

**Campos de datos**

|          |                |  |
|----------|----------------|--|
| uint32_t | ssel2_n: 1     |  |
| uint32_t | ssel3_n: 1     |  |
| uint32_t | eot: 1         |  |
| uint32_t | eof: 1         |  |
| uint32_t | rxignore: 1    |  |
| uint32_t | __pad0__: 1    |  |
| uint32_t | data_length: 4 |  |
| uint32_t | __pad1__: 4    |  |

**7.7.3. Documentación de las funciones****7.7.3.1. hal\_spi\_master\_mode\_init()**

```
void hal_spi_master_mode_init (
    hal_spi_sel_en inst,
    const hal_spi_master_mode_config_t * config )
```

Inicializar SPI en modo master.

**Parámetros**

|    |               |                                |
|----|---------------|--------------------------------|
| in | <i>inst</i>   | Instancia de SPI a inicializar |
| in | <i>config</i> | Configuración deseada          |

**7.7.3.2. hal\_spi\_master\_mode\_rx\_data()**

```
uint16_t hal_spi_master_mode_rx_data (
    hal_spi_sel_en inst )
```

Leer el dato recibido.

**Parámetros**

|    |             |                       |
|----|-------------|-----------------------|
| in | <i>inst</i> | Instancia a consultar |
|----|-------------|-----------------------|

**Devuelve**

Dato recibido



### 7.7.3.3. hal\_spi\_master\_mode\_config\_tx()

```
void hal_spi_master_mode_config_tx (
    hal_spi_sel_en inst,
    const hal_spi_master_mode_tx_config_t * config )
```

Configurar la transmision.

#### Parámetros

|    |               |                                           |
|----|---------------|-------------------------------------------|
| in | <i>inst</i>   | Instancia a configurar                    |
| in | <i>config</i> | Configuracion para la transmision deseada |

### 7.7.3.4. hal\_spi\_master\_mode\_tx\_data()

```
void hal_spi_master_mode_tx_data (
    hal_spi_sel_en inst,
    const hal_spi_master_mode_tx_data_t * data )
```

Transmitir dato.

#### Parámetros

|    |             |                                            |
|----|-------------|--------------------------------------------|
| in | <i>inst</i> | Instancia a utilizar                       |
| in | <i>data</i> | Dato a transmitir, con controles asociados |

### 7.7.3.5. hal\_spi\_master\_mode\_register\_tx\_callback()

```
void hal_spi_master_mode_register_tx_callback (
    hal_spi_sel_en inst,
    void(*) (void) new_callback )
```

Actualizar callback en TXRDY.

#### Parámetros

|    |                     |                                    |
|----|---------------------|------------------------------------|
| in | <i>inst</i>         | Instancia a configurar             |
| in | <i>new_callback</i> | Nuevo callback a ejecutar en TXRDY |

### 7.7.3.6. hal\_spi\_master\_mode\_register\_rx\_callback()

```
void hal_spi_master_mode_register_rx_callback (
```

```
hal_spi_sel_en inst,  
void(*) (void) new_callback )
```

Actualizar callback en RXRDY.

#### Parámetros

|    |                     |                                    |
|----|---------------------|------------------------------------|
| in | <i>inst</i>         | Instancia a configurar             |
| in | <i>new_callback</i> | Nuevo callback a ejecutar en RXRDY |

## 7.8. Referencia del Archivo includes/hal/HAL\_SYSCON.h

Declaraciones a nivel de aplicacion del perifero SYSCON (LPC845)

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

Dependencia gráfica adjunta para HAL\_SYSCON.h:

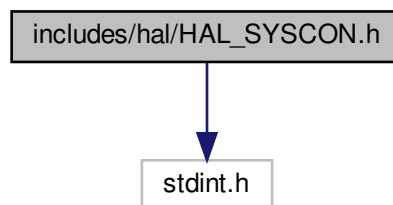
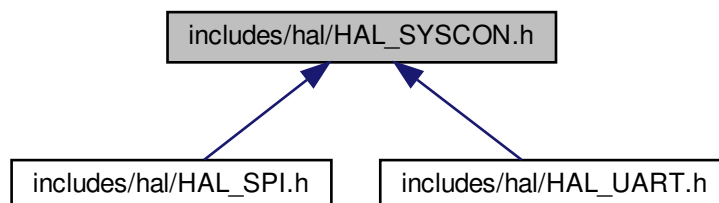


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Enumeraciones

- enum `hal_syscon_clkout_source_sel_en` {  
`HAL_SYSCON_CLKOUT_SOURCE_SEL_FRO` = 0, `HAL_SYSCON_CLKOUT_SOURCE_SEL_MAIN_C`  
`CLOCK`, `HAL_SYSCON_CLKOUT_SOURCE_SEL_SYS_PLL`, `HAL_SYSCON_CLKOUT_SOURCE_SE`  
`L_EXT_CLOCK`,  
`HAL_SYSCON_CLKOUT_SOURCE_SEL_WATCHDOG_OSC` }
- enum `hal_syscon_frg_clock_sel_en` { `HAL_SYSCON_FRG_CLOCK_SEL_FRO` = 0, `HAL_SYSCON_`  
`FRG_CLOCK_SEL_MAIN_CLOCK`, `HAL_SYSCON_FRG_CLOCK_SEL_SYS_PLL`, `HAL_SYSCON_FR`  
`G_CLOCK_SEL_NONE` }
- enum `hal_syscon_peripheral_sel_en` {  
`HAL_SYSCON_PERIPHERAL_SEL_UART0` = 0, `HAL_SYSCON_PERIPHERAL_SEL_UART1`, `HAL_S`  
`YSCON_PERIPHERAL_SEL_UART2`, `HAL_SYSCON_PERIPHERAL_SEL_UART3`,  
`HAL_SYSCON_PERIPHERAL_SEL_UART4`, `HAL_SYSCON_PERIPHERAL_SEL_IIC0`, `HAL_SYSCON_`  
`_PERIPHERAL_SEL_IIC1`, `HAL_SYSCON_PERIPHERAL_SEL_IIC2`,  
`HAL_SYSCON_PERIPHERAL_SEL_IIC3`, `HAL_SYSCON_PERIPHERAL_SEL_SPI0`, `HAL_SYSCON_`  
`PERIPHERAL_SEL_SPI1` }
- enum `hal_syscon_peripheral_clock_sel_en` {  
`HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRO` = 0, `HAL_SYSCON_PERIPHERAL_CLOCK_SEL_`  
`MAIN`, `HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRG0`, `HAL_SYSCON_PERIPHERAL_CLOCK_S`  
`EL_FRG1`,  
`HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRO_DIV`, `HAL_SYSCON_PERIPHERAL_CLOCK_SEL_`  
`_NONE` = 7 }
- enum `hal_syscon_iocon_glitch_sel_en` {  
`HAL_SYSCON_IOCON_GLITCH_SEL_0` = 0, `HAL_SYSCON_IOCON_GLITCH_SEL_1`, `HAL_SYSCON_`  
`_IOCON_GLITCH_SEL_2`, `HAL_SYSCON_IOCON_GLITCH_SEL_3`,  
`HAL_SYSCON_IOCON_GLITCH_SEL_4`, `HAL_SYSCON_IOCON_GLITCH_SEL_5`, `HAL_SYSCON_IO`  
`CON_GLITCH_SEL_6`, `HAL_SYSCON_IOCON_GLITCH_SEL_7` }
- enum `hal_syscon_pll_source_sel_en` { `HAL_SYSCON_PLL_SOURCE_SEL_FRO` = 0, `HAL_SYSCO`  
`N_PLL_SOURCE_SEL_EXT_CLK`, `HAL_SYSCON_PLL_SOURCE_SEL_WATCHDOG`, `HAL_SYSCON_`  
`_PLL_SOURCE_SEL_FRO_DIV` }

## Funciones

- `uint32_t hal_syscon_get_system_clock` (void)  
*Obtener la frecuencia actual del main clock.*
- `uint32_t hal_syscon_get_fro_clock` (void)  
*Obtener la frecuencia actual del FRO.*
- void `hal_syscon_config_external_crystal` (uint32\_t crystal\_freq, uint8\_t use\_as\_main)  
*Configurar el ext clock a partir de un cristal externo.*
- void `hal_syscon_config_fro_direct` (uint8\_t direct, uint8\_t use\_as\_main)  
*Configurar el clock FRO.*
- void `hal_syscon_config_clkout` (uint8\_t port, uint8\_t pin, hal\_syscon\_clkout\_source\_sel\_en clock\_source,  
uint8\_t divider)  
*Configurar el pin de clock out (salida de clock hacia afuera)*
- void `hal_syscon_config_frg` (uint8\_t inst, hal\_syscon\_frg\_clock\_sel\_en clock\_source, uint32\_t mul)  
*Configurar el divisor fraccional.*
- void `hal_syscon_set_peripheral_clock_source` (hal\_syscon\_peripheral\_sel\_en peripheral, hal\_syscon\_↵  
peripheral\_clock\_sel\_en clock\_source)  
*Fijar la fuente de clock de un periférico.*
- `uint32_t hal_syscon_get_peripheral_clock` (hal\_syscon\_peripheral\_sel\_en peripheral)  
*Obtener la frecuencia de clock en Hz configurada para cierto periférico.*
- void `hal_syscon_set_iocon_glitch_divider` (hal\_syscon\_iocon\_glitch\_sel\_en sel, uint32\_t div)  
*Configurar divisor para el clock de glitches del IOCON.*

- void [hal\\_syscon\\_config\\_pll](#) (hal\_syscon\_pll\_source\_sel\_en clock\_source, uint32\_t freq)  
*Configurar el PLL.*
- uint32\_t [hal\\_syscon\\_get\\_pll\\_clock](#) (void)  
*Obtener frecuencia actual configurada del PLL.*

### 7.8.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico SYSCON (LPC845)

#### Autor

Augusto Santini

#### Fecha

6/2019

#### Versión

1.0

### 7.8.2. Documentación de las funciones

#### 7.8.2.1. [hal\\_syscon\\_get\\_system\\_clock\(\)](#)

```
uint32_t hal_syscon_get_system_clock (  
    void )
```

Obtener la frecuencia actual del main clock.

#### Devuelve

Frecuencia del main clock en Hz

#### 7.8.2.2. [hal\\_syscon\\_get\\_fro\\_clock\(\)](#)

```
uint32_t hal_syscon_get_fro_clock (  
    void )
```

Obtener la frecuencia actual del FRO.

#### Devuelve

Frecuencia del FRO en Hz

#### 7.8.2.3. [hal\\_syscon\\_config\\_external\\_crystal\(\)](#)

```
void hal_syscon_config_external_crystal (  
    uint32_t crystal_freq,  
    uint8_t use_as_main )
```

Configurar el ext clock a partir de un cristal externo.

**Parámetros**

|    |                     |                                                                             |
|----|---------------------|-----------------------------------------------------------------------------|
| in | <i>crystal_freq</i> | Frecuencia del cristal externo utilizado                                    |
| in | <i>use_as_main</i>  | Si es distinto de cero, se utilizara el oscilador a cristal como main clock |

**7.8.2.4. hal\_syscon\_config\_fro\_direct()**

```
void hal_syscon_config_fro_direct (
    uint8_t direct,
    uint8_t use_as_main )
```

Configurar el clock FRO.

**Parámetros**

|    |                    |                                                             |
|----|--------------------|-------------------------------------------------------------|
| in | <i>direct</i>      | Si es distinto de cero se omite el divisor del FRO          |
| in | <i>use_as_main</i> | Si es distinto de cero, se utilizara el FRO como main clock |

**7.8.2.5. hal\_syscon\_config\_clkout()**

```
void hal_syscon_config_clkout (
    uint8_t port,
    uint8_t pin,
    hal_syscon_clkout_source_sel_en clock_source,
    uint8_t divider )
```

Configurar el pin de clock out (salida de clock hacia afuera)

**Parámetros**

|    |                     |                                               |
|----|---------------------|-----------------------------------------------|
| in | <i>port</i>         | Numero de puerto por donde sacar el clock out |
| in | <i>pin</i>          | Numero de pin por donde sacar el clock out    |
| in | <i>clock_source</i> | Fuente deseada para la salida clock out       |
| in | <i>divider</i>      | Divisor deseado para la salida clock out      |

**7.8.2.6. hal\_syscon\_config\_frg()**

```
void hal_syscon_config_frg (
    uint8_t inst,
    hal_syscon_frg_clock_sel_en clock_source,
    uint32_t mul )
```

Configurar el divisor fraccional.

El divisor siempre se debe fijar en 256 para estos MCU.

**Parámetros**

|    |                     |                                        |
|----|---------------------|----------------------------------------|
| in | <i>inst</i>         | Instancia de FRG a configurar          |
| in | <i>clock_source</i> | Fuente de clock de entrada para el FRG |
| in | <i>mul</i>          | Multiplificador deseado                |

**7.8.2.7. hal\_syscon\_set\_peripheral\_clock\_source()**

```
void hal_syscon_set_peripheral_clock_source (
    hal_syscon_peripheral_sel_en peripheral,
    hal_syscon_peripheral_clock_sel_en clock_source )
```

Fijar la fuente de clock de un periférico.

**Parámetros**

|    |                     |                         |
|----|---------------------|-------------------------|
| in | <i>peripheral</i>   | Periférico deseado      |
| in | <i>clock_source</i> | Fuente de clock deseada |

**7.8.2.8. hal\_syscon\_get\_peripheral\_clock()**

```
uint32_t hal_syscon_get_peripheral_clock (
    hal_syscon_peripheral_sel_en peripheral )
```

Obtener la frecuencia de clock en Hz configurada para cierto periférico.

**Parámetros**

|    |                   |                    |
|----|-------------------|--------------------|
| in | <i>peripheral</i> | Periférico deseado |
|----|-------------------|--------------------|

**Devuelve**

Frecuencia en Hz del clock del periférico

**7.8.2.9. hal\_syscon\_set\_iocon\_glitch\_divider()**

```
void hal_syscon_set_iocon_glitch_divider (
    hal_syscon_iocon_glitch_sel_en sel,
    uint32_t div )
```

Configurar divisor para el clock de glitches del IOCON.

**Parámetros**

|    |            |                           |
|----|------------|---------------------------|
| in | <i>sel</i> | Selección de divisor      |
| in | <i>div</i> | Valor de división deseado |

**7.8.2.10. hal\_syscon\_config\_pll()**

```
void hal_syscon_config_pll (
    hal_syscon_pll_source_sel_en clock_source,
    uint32_t freq )
```

Configurar el PLL.

**Parámetros**

|    |                     |                                           |
|----|---------------------|-------------------------------------------|
| in | <i>clock_source</i> | Fuente de clock de referencia para el PLL |
| in | <i>freq</i>         | Frecuencia deseada de salida del PLL      |

**7.8.2.11. hal\_syscon\_get\_pll\_clock()**

```
uint32_t hal_syscon_get_pll_clock (
    void )
```

Obtener frecuencia actual configurada del PLL.

**Devuelve**

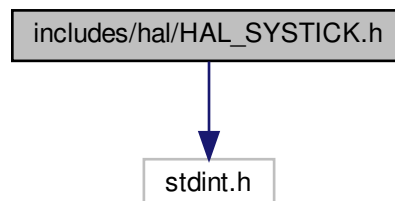
Frecuencia actual del PLL en Hz

**7.9. Referencia del Archivo includes/hal/HAL\_SYSTICK.h**

Declaraciones a nivel de aplicación del periférico SYSTICK (LPC845)

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

Dependencia gráfica adjunta para HAL\_SYSTICK.h:





## Funciones

- void [hal\\_systick\\_init](#) (uint32\_t tick\_us, void(\*callback)(void))  
*Inicializacion del SYSTICK.*
- void [hal\\_systick\\_update\\_callback](#) (void(\*callback)(void))  
*Actualizar callback del SYSTICK.*

### 7.9.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico SYSTICK (LPC845)

#### Autor

Augusto Santini

#### Fecha

3/2020

#### Versión

1.0

### 7.9.2. Documentación de las funciones

#### 7.9.2.1. hal\_systick\_init()

```
void hal_systick_init (
    uint32_t tick_us,
    void(*) (void) callback )
```

Inicializacion del SYSTICK.

#### Parámetros

|    |                 |                                              |
|----|-----------------|----------------------------------------------|
| in | <i>tick_us</i>  | Tiempo en microsegundos deseado para el tick |
| in | <i>callback</i> | Funcion a llamar en cada tick                |

#### Ejemplos:

[Ejemplo\\_ADC.c.](#)

### 7.9.2.2. hal\_systick\_update\_callback()

```
void hal_systick_update_callback (
    void(*) (void) callback )
```

Actualizar callback del SYSTICK.

#### Parámetros

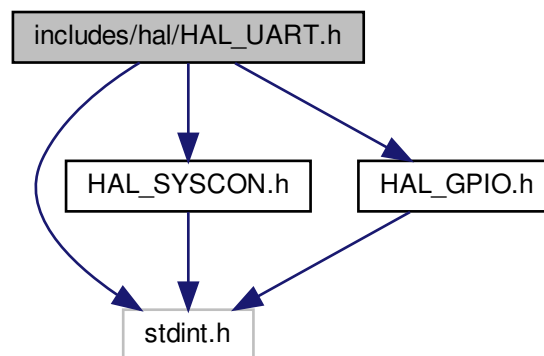
|    |                 |                                        |
|----|-----------------|----------------------------------------|
| in | <i>callback</i> | Nuevo callback a ejecutar en cada tick |
|----|-----------------|----------------------------------------|

## 7.10. Referencia del Archivo includes/hal/HAL\_UART.h

Declaraciones a nivel de aplicacion del periferico UART (LPC845)

```
#include <stdint.h>
#include <HAL_SYSCON.h>
#include <HAL_GPIO.h>
```

Dependencia gráfica adjunta para HAL\_UART.h:



### Estructuras de datos

- struct `hal_uart_config_t`

### Enumeraciones

- enum `hal_uart_data_len_en` { `HAL_UART_DATALEN_7BIT` = 0, `HAL_UART_DATALEN_8BIT`, `HAL_UART_DATALEN_9BIT` }
- enum `hal_uart_parity_en` { `HAL_UART_PARITY_NO_PARITY` = 0, `HAL_UART_PARITY_EVEN` = 2, `HAL_UART_PARITY_ODD` }
- enum `hal_uart_stop_en` { `HAL_UART_STOPLEN_1BIT` = 0, `HAL_UART_STOPLEN_2BIT` }

- enum `hal_uart_oversampling_en` {  
`HAL_UART_OVERSAMPLING_X5 = 4`, `HAL_UART_OVERSAMPLING_X6`, `HAL_UART_OVERSAMPLING_X7`, `HAL_UART_OVERSAMPLING_X8`,  
`HAL_UART_OVERSAMPLING_X9`, `HAL_UART_OVERSAMPLING_X10`, `HAL_UART_OVERSAMPLING_X11`, `HAL_UART_OVERSAMPLING_X12`,  
`HAL_UART_OVERSAMPLING_X13`, `HAL_UART_OVERSAMPLING_X14`, `HAL_UART_OVERSAMPLING_X15`, `HAL_UART_OVERSAMPLING_X16` }
- enum `hal_uart_tx_result` { `HAL_UART_TX_RESULT_OK = 0`, `HAL_UART_TX_RESULT_NOT_READY` }
- enum `hal_uart_rx_result` { `HAL_UART_RX_RESULT_OK = 0`, `HAL_UART_RX_RESULT_NOT_READY` }

## Funciones

- void `hal_uart_init` (uint8\_t inst, const `hal_uart_config_t` \*config)  
*Inicializar UART con los parametros deseados.*
- hal\_uart\_tx\_result `hal_uart_tx_byte` (uint8\_t inst, uint32\_t data)  
*Transmitir un dato mediante la UART.*
- hal\_uart\_rx\_result `hal_uart_rx_byte` (uint8\_t inst, uint32\_t \*data)  
*Recibir un dato de la UART.*
- void `hal_uart_register_tx_callback` (uint8\_t inst, void(\*new\_callback)(void))  
*Registrar el callback a ser llamado una vez finalizada la transmision de un dato por UART.*
- void `hal_uart_register_rx_callback` (uint8\_t inst, void(\*new\_callback)(void))  
*Registrar el callback a ser llamado en la recepcion de un dato por UART.*
- void `UART3_irq` (void)  
*Interrupcion de UART3.*
- void `UART4_irq` (void)  
*Interrupcion de UART4.*

### 7.10.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico UART (LPC845)

#### Autor

Augusto Santini

#### Fecha

3/2020

#### Versión

1.0

### 7.10.2. Documentación de las funciones

#### 7.10.2.1. `hal_uart_init()`

```
void hal_uart_init (
    uint8_t inst,
    const hal_uart_config_t * config )
```

Inicializar UART con los parametros deseados.

**Parámetros**

|    |               |                                    |
|----|---------------|------------------------------------|
| in | <i>inst</i>   | Que instancia de UART inicializar  |
| in | <i>config</i> | Puntero a configuracion de la UART |

**7.10.2.2. hal\_uart\_tx\_byte()**

```
hal_uart_tx_result hal_uart_tx_byte (
    uint8_t inst,
    uint32_t data )
```

Transmitir un dato mediante la UART.

**Parámetros**

|    |             |                                               |
|----|-------------|-----------------------------------------------|
| in | <i>inst</i> | Que instancia de UART usar                    |
| in | <i>data</i> | Dato a transmitir. Puede ser de 7, 8 o 9 bits |

**7.10.2.3. hal\_uart\_rx\_byte()**

```
hal_uart_rx_result hal_uart_rx_byte (
    uint8_t inst,
    uint32_t * data )
```

Recibir un dato de la UART.

**Parámetros**

|    |             |                                          |
|----|-------------|------------------------------------------|
| in | <i>inst</i> | Que instancia de UART usar               |
| in | <i>data</i> | Puntero a donde guardar el dato recibido |

**Devuelve**

Estado de la recepcion

**7.10.2.4. hal\_uart\_register\_tx\_callback()**

```
void hal_uart_register_tx_callback (
    uint8_t inst,
    void(*) (void) new_callback )
```

Registrar el callback a ser llamado una vez finalizada la transmision de un dato por UART.

## Parámetros

|    |                     |                                                                               |
|----|---------------------|-------------------------------------------------------------------------------|
| in | <i>inst</i>         | A que instancia de UART registrar el callback                                 |
| in | <i>new_callback</i> | Puntero a funcion a llamar cada vez que se termina de enviar un dato por UART |

7.10.2.5. `hal_uart_register_rx_callback()`

```
void hal_uart_register_rx_callback (
    uint8_t inst,
    void(*) (void) new_callback )
```

Registrar el callback a ser llamado en la recepcion de un dato por UART.

## Parámetros

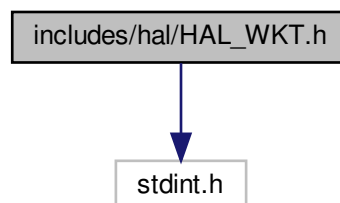
|    |                     |                                                                    |
|----|---------------------|--------------------------------------------------------------------|
| in | <i>inst</i>         | A que instancia de UART registrar el callback                      |
| in | <i>new_callback</i> | Puntero a funcion a llamar cada vez que se recibe un dato por UART |

## 7.11. Referencia del Archivo includes/hal/HAL\_WKT.h

Declaraciones a nivel de aplicacion del periferico WKT (LPC845)

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

Dependencia gráfica adjunta para HAL\_WKT.h:



## Enumeraciones

- enum `hal_wkt_clock_source_en` { `HAL_WKT_CLOCK_SOURCE_FRO_DIV = 0`, `HAL_WKT_CLOCK_SOURCE_LOW_POWER_OSC`, `HAL_WKT_CLOCK_SOURCE_EXTERNAL` }

## Funciones

- void `hal_wkt_init` (`hal_wkt_clock_source_en` `clock_sel`, `uint32_t` `ext_clock_value`, `void(*callback)(void)`)  
*Inicializar el WKT.*
- void `hal_wkt_select_clock_source` (`hal_wkt_clock_source_en` `clock_sel`, `uint32_t` `ext_clock_value`)
- void `hal_wkt_register_callback` (`void(*new_callback)(void)`)  
*Registrar un callback para la interrupcion del WKT.*
- void `hal_wkt_start_count` (`uint32_t` `time_useg`)
- void `hal_wkt_start_count_with_value` (`uint32_t` `value`)

### 7.11.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico WKT (LPC845)

#### Autor

Augusto Santini

#### Fecha

3/2020

#### Versión

1.0

### 7.11.2. Documentación de las funciones

#### 7.11.2.1. `hal_wkt_init()`

```
void hal_wkt_init (
    hal_wkt_clock_source_en clock_sel,
    uint32_t ext_clock_value,
    void(*) (void) callback )
```

Inicializar el WKT.

#### Parámetros

|    |                        |                                                                                |
|----|------------------------|--------------------------------------------------------------------------------|
| in | <i>clock_sel</i>       | Seleccin de clock deseada para el WKT                                          |
| in | <i>ext_clock_value</i> | Valor de clock externo (si la seleccion es interna, no importa este parametro) |
| in | <i>callback</i>        | Callback a ejecutar en la interrupcion del WKT                                 |

### 7.11.2.2. hal\_wkt\_register\_callback()

```
void hal_wkt_register_callback (
    void(*) (void) new_callback )
```

Registrar un callback para la interrupcion del WKT.

#### Parámetros

|    |                     |                                             |
|----|---------------------|---------------------------------------------|
| in | <i>new_callback</i> | Nuevo callback para la interrupcion del WKT |
|----|---------------------|---------------------------------------------|





## Capítulo 8

# Documentación de ejemplos

### 8.1. Ejemplo\_ADC.c

Ejemplo sobre la utilización del *ADC*. El programa utiliza el clock por default con el que comienza el microcontrolador, es decir, el *Free Running Oscillator* funcionando a  $12MHz$ .

El periférico será configurado con las siguientes características:

- Funcionamiento **sincrónico**
- Frecuencia de muestreo de  $1MHz$
- Modo bajo consumo inhabilitado

La secuencia A es configurada para generar conversiones en los canales 0 y 8:

- El canal 0 está conectado al preset propio del stick de desarrollo (Puerto 0 pin 7)
- El canal 8 está ubicado en el pin número 3 (Puerto 0 pin 18) y se le puede conectar un preset externo entre VDD y GND.

Además, la secuencia tendrá la siguiente configuración:

- Trigger: Únicamente se disparan conversiones por software
- Bypass sincronismo: Sí
- Modo de interrupción: Cuando termina la secuencia completa
- Burst: Inhabilitado
- Un trigger dispara: Una conversión de secuencia completa
- Secuencia A como baja prioridad: No

Una vez inicializado el periférico, se configura el periférico *Systick* para interrumpir cada  $1mseg$  y mediante su manejador se lleva la cuenta de los milisegundos transcurridos. Una vez transcurridos  $1000mseg$ , se dispara una conversión de *ADC*, y sus resultados se guardan en dos variables globales.

Ubicando un *breakpoint* adecuadamente, se pueden leer los resultados de las conversiones ya ubicadas en las variables globales.

```

#include <cr_section_macros.h>
#include <stddef.h>
#include <HAL_ADC.h>
#include <HAL_SYSTICK.h>

/* Máscara de configuración de canales habilitados para la secuencia a configurar */
#define ADC_CHANNELS ((1 << 0) | (1 << 8))

/* Macro para definir el tiempo de interrupción del \e Systick en \b microsegundos */
#define TICK_TIME_USEG (1000)

/* Frecuencia de muestreo a utilizar por el ADC */
#define ADC_SAMPLE_FREQ (1000000)

#define ADC_SEQUENCE (HAL_ADC_SEQUENCE_SEL_A)

/* Tiempo de disparo de conversiones de \e ADC en \b milisegundos */
#define ADC_CONVERSION_TIME_MSEG (1000)

static void adc_callback(void);

static void systick_callback(void);

static uint8_t flag_secuencia_adc_completada = 0; /* Flag para indicar finalización de secuencia de
conversión de \e ADC */

/*
 * Variables para guardar los resultados de la secuencia de conversión
 */
static hal_adc_sequence_result_t resultados_conversion_adc[2];

/* Configuración de la secuencia. Como no va a cambiar es declarada \e const */
static const hal_adc_sequence_config_t adc_config =
{
    .channels = ADC_CHANNELS,
    .trigger = HAL_ADC_TRIGGER_SEL_NONE,
    .trigger_pol = HAL_ADC_TRIGGER_POL_SEL_NEGATIVE_EDGE,
    .sync_bypass = HAL_ADC_SYNC_SEL_BYPASS_SYNC,
    .mode = HAL_ADC_INTERRUPT_MODE_EOS,
    .burst = 0,
    .single_step = 0,
    .low_priority = 0,
    .callback = adc_callback
};

/*
 * @brief Punto de entrada del programa
 * @return Nunca debería terminar esta función
 */
int main(void)
{
    // Inicialización del periférico en modo SINCRÓNICO
    hal_adc_init_sync_mode(ADC_SAMPLE_FREQ,
        HAL_ADC_LOW_POWER_MODE_DISABLED);

    // Configuración de la secuencia a utilizar
    hal_adc_config_sequence(ADC_SEQUENCE, &adc_config);

    // Inicialización del \e Systick con el tiempo de tick adecuado
    hal_systick_init(TICK_TIME_USEG, systick_callback);

    while(1)
    {
        if(flag_secuencia_adc_completada == 1) // Esto o hacer "if(flag_secuencia_adc_completada)" es
            indistinto
        {
            uint32_t variable_auxiliar;

            flag_secuencia_adc_completada = 0;

            (void) variable_auxiliar; // Esta línea es ideal para colocar el breakpoint!
        }
    }

    return 0;
}

/*
 * @brief Callback a ejecutar en cada tick del \e Systick
 */
static void systick_callback(void)
{
    static uint32_t contador_disparo_adc = 0;

    // Conteo con valor límite
    contador_disparo_adc = (contador_disparo_adc + 1) % ADC_CONVERSION_TIME_MSEG;
}

```

```
    if(contador_disparo_adc == 0) // Esto o hacer "if(!contador_disparo_adc)" es indistinto
    {
        hal_adc_start_sequence(ADC_SEQUENCE);
    }
}

/*
 * @brief Callback a ejecutar en cada finalización de conversión de \b secuencia de \e ADC
 */
static void adc_callback(void)
{
    // Obtención de resultados de conversión
    hal_adc_get_sequence_result(ADC_SEQUENCE, resultados_conversion_adc);

    flag_secuencia_adc_completada = 1;
}
```



# Índice alfabético

## ADC, 9

- hal\_adc\_clock\_source\_en, 12
- hal\_adc\_config\_sequence, 16
- hal\_adc\_enable\_sequence, 16
- hal\_adc\_get\_sequence\_result, 17
- hal\_adc\_init\_async\_mode, 15
- hal\_adc\_init\_sync\_mode, 15
- hal\_adc\_interrupt\_mode\_en, 13
- hal\_adc\_low\_power\_mode\_en, 12
- hal\_adc\_result\_channel\_en, 14
- hal\_adc\_sequence\_result\_en, 14
- hal\_adc\_sequence\_sel\_en, 12
- hal\_adc\_start\_sequence, 16
- hal\_adc\_sync\_sel\_en, 13
- hal\_adc\_trigger\_pol\_sel\_en, 13
- hal\_adc\_trigger\_sel\_en, 12

## callback

- hal\_adc\_sequence\_config\_t, 21

## channels

- hal\_adc\_sequence\_config\_t, 19

## HAL\_CTIMER.h

- hal\_ctimer\_pwm\_mode\_config\_channel, 29
- hal\_ctimer\_pwm\_mode\_init, 29
- hal\_ctimer\_pwm\_mode\_set\_period, 29
- hal\_ctimer\_timer\_mode\_config\_match, 29
- hal\_ctimer\_timer\_mode\_init, 28

## HAL\_DAC.h

- hal\_dac\_init, 31

## HAL\_GPIO.h

- hal\_gpio\_clear\_pin, 34
- hal\_gpio\_init, 33
- hal\_gpio\_read\_pin, 35
- hal\_gpio\_set\_dir, 34
- hal\_gpio\_set\_pin, 34
- hal\_gpio\_toggle\_pin, 35

## HAL\_IOCON.h

- hal\_iocon\_config\_io, 37

## HAL\_PININT.h

- hal\_pinint\_configure\_pin\_interrupt, 39
- hal\_pinint\_register\_callback, 39

## HAL\_SPI.h

- hal\_spi\_master\_mode\_config\_tx, 42
- hal\_spi\_master\_mode\_init, 42
- hal\_spi\_master\_mode\_register\_rx\_callback, 43
- hal\_spi\_master\_mode\_register\_tx\_callback, 43
- hal\_spi\_master\_mode\_rx\_data, 42
- hal\_spi\_master\_mode\_tx\_data, 43

## HAL\_SYSCON.h

- hal\_syscon\_config\_clkout, 47
- hal\_syscon\_config\_external\_crystal, 46
- hal\_syscon\_config\_frg, 47
- hal\_syscon\_config\_fro\_direct, 47
- hal\_syscon\_config\_pll, 50
- hal\_syscon\_get\_fro\_clock, 46
- hal\_syscon\_get\_peripheral\_clock, 49
- hal\_syscon\_get\_pll\_clock, 50
- hal\_syscon\_get\_system\_clock, 46
- hal\_syscon\_set\_iocon\_glitch\_divider, 49
- hal\_syscon\_set\_peripheral\_clock\_source, 49

## HAL\_SYSTICK.h

- hal\_systick\_init, 51
- hal\_systick\_update\_callback, 51

## HAL\_UART.h

- hal\_uart\_init, 53
- hal\_uart\_register\_rx\_callback, 55
- hal\_uart\_register\_tx\_callback, 54
- hal\_uart\_rx\_byte, 54
- hal\_uart\_tx\_byte, 54

## HAL\_WKT.h

- hal\_wkt\_init, 56
- hal\_wkt\_register\_callback, 56

## hal\_adc\_clock\_source\_en

- ADC, 12

## hal\_adc\_config\_sequence

- ADC, 16

## hal\_adc\_enable\_sequence

- ADC, 16

## hal\_adc\_get\_sequence\_result

- ADC, 17

## hal\_adc\_init\_async\_mode

- ADC, 15

## hal\_adc\_init\_sync\_mode

- ADC, 15

## hal\_adc\_interrupt\_mode\_en

- ADC, 13

## hal\_adc\_low\_power\_mode\_en

- ADC, 12

## hal\_adc\_result\_channel\_en

- ADC, 14

## hal\_adc\_sequence\_config\_t, 19

- callback, 21

- channels, 19

- low\_priority, 20

- mode, 20

- single\_step, 20

- sync\_bypass, 20

- trigger, 20

trigger\_pol, [20](#)  
 hal\_adc\_sequence\_result\_en  
     ADC, [14](#)  
 hal\_adc\_sequence\_result\_t, [11](#)  
 hal\_adc\_sequence\_sel\_en  
     ADC, [12](#)  
 hal\_adc\_start\_sequence  
     ADC, [16](#)  
 hal\_adc\_sync\_sel\_en  
     ADC, [13](#)  
 hal\_adc\_trigger\_pol\_sel\_en  
     ADC, [13](#)  
 hal\_adc\_trigger\_sel\_en  
     ADC, [12](#)  
 hal\_ctimer\_match\_config\_t, [21](#)  
 hal\_ctimer\_pwm\_channel\_config\_t, [21](#)  
 hal\_ctimer\_pwm\_config\_t, [22](#)  
 hal\_ctimer\_pwm\_mode\_config\_channel  
     HAL\_CTIMER.h, [29](#)  
 hal\_ctimer\_pwm\_mode\_init  
     HAL\_CTIMER.h, [29](#)  
 hal\_ctimer\_pwm\_mode\_set\_period  
     HAL\_CTIMER.h, [29](#)  
 hal\_ctimer\_timer\_mode\_config\_match  
     HAL\_CTIMER.h, [29](#)  
 hal\_ctimer\_timer\_mode\_init  
     HAL\_CTIMER.h, [28](#)  
 hal\_dac\_ctrl\_config\_t, [31](#)  
 hal\_dac\_init  
     HAL\_DAC.h, [31](#)  
 hal\_gpio\_clear\_pin  
     HAL\_GPIO.h, [34](#)  
 hal\_gpio\_init  
     HAL\_GPIO.h, [33](#)  
 hal\_gpio\_read\_pin  
     HAL\_GPIO.h, [35](#)  
 hal\_gpio\_set\_dir  
     HAL\_GPIO.h, [34](#)  
 hal\_gpio\_set\_pin  
     HAL\_GPIO.h, [34](#)  
 hal\_gpio\_toggle\_pin  
     HAL\_GPIO.h, [35](#)  
 hal\_iocon\_config\_io  
     HAL\_IOCON.h, [37](#)  
 hal\_iocon\_config\_t, [37](#)  
 hal\_pinint\_config\_t, [22](#)  
 hal\_pinint\_configure\_pin\_interrupt  
     HAL\_PININT.h, [39](#)  
 hal\_pinint\_register\_callback  
     HAL\_PININT.h, [39](#)  
 hal\_spi\_master\_mode\_config\_t, [22](#)  
 hal\_spi\_master\_mode\_config\_tx  
     HAL\_SPI.h, [42](#)  
 hal\_spi\_master\_mode\_init  
     HAL\_SPI.h, [42](#)  
 hal\_spi\_master\_mode\_register\_rx\_callback  
     HAL\_SPI.h, [43](#)  
 hal\_spi\_master\_mode\_register\_tx\_callback  
     HAL\_SPI.h, [43](#)  
 hal\_spi\_master\_mode\_rx\_data  
     HAL\_SPI.h, [42](#)  
 hal\_spi\_master\_mode\_tx\_config\_t, [41](#)  
 hal\_spi\_master\_mode\_tx\_data  
     HAL\_SPI.h, [43](#)  
 hal\_spi\_master\_mode\_tx\_data\_t, [41](#)  
 hal\_syscon\_config\_clkout  
     HAL\_SYSCON.h, [47](#)  
 hal\_syscon\_config\_external\_crystal  
     HAL\_SYSCON.h, [46](#)  
 hal\_syscon\_config\_frg  
     HAL\_SYSCON.h, [47](#)  
 hal\_syscon\_config\_fro\_direct  
     HAL\_SYSCON.h, [47](#)  
 hal\_syscon\_config\_pll  
     HAL\_SYSCON.h, [50](#)  
 hal\_syscon\_get\_fro\_clock  
     HAL\_SYSCON.h, [46](#)  
 hal\_syscon\_get\_peripheral\_clock  
     HAL\_SYSCON.h, [49](#)  
 hal\_syscon\_get\_pll\_clock  
     HAL\_SYSCON.h, [50](#)  
 hal\_syscon\_get\_system\_clock  
     HAL\_SYSCON.h, [46](#)  
 hal\_syscon\_set\_iocon\_glitch\_divider  
     HAL\_SYSCON.h, [49](#)  
 hal\_syscon\_set\_peripheral\_clock\_source  
     HAL\_SYSCON.h, [49](#)  
 hal\_systick\_init  
     HAL\_SYSTICK.h, [51](#)  
 hal\_systick\_update\_callback  
     HAL\_SYSTICK.h, [51](#)  
 hal\_uart\_config\_t, [23](#)  
 hal\_uart\_init  
     HAL\_UART.h, [53](#)  
 hal\_uart\_register\_rx\_callback  
     HAL\_UART.h, [55](#)  
 hal\_uart\_register\_tx\_callback  
     HAL\_UART.h, [54](#)  
 hal\_uart\_rx\_byte  
     HAL\_UART.h, [54](#)  
 hal\_uart\_tx\_byte  
     HAL\_UART.h, [54](#)  
 hal\_wkt\_init  
     HAL\_WKT.h, [56](#)  
 hal\_wkt\_register\_callback  
     HAL\_WKT.h, [56](#)  
 includes/hal/HAL\_ADC.h, [25](#)  
 includes/hal/HAL\_CTIMER.h, [27](#)  
 includes/hal/HAL\_DAC.h, [30](#)  
 includes/hal/HAL\_GPIO.h, [32](#)  
 includes/hal/HAL\_IOCON.h, [35](#)  
 includes/hal/HAL\_PININT.h, [37](#)  
 includes/hal/HAL\_SPI.h, [40](#)  
 includes/hal/HAL\_SYSCON.h, [44](#)  
 includes/hal/HAL\_SYSTICK.h, [50](#)  
 includes/hal/HAL\_UART.h, [52](#)

includes/hal/HAL\_WKT.h, [55](#)

low\_priority  
    hal\_adc\_sequence\_config\_t, [20](#)

mode  
    hal\_adc\_sequence\_config\_t, [20](#)

single\_step  
    hal\_adc\_sequence\_config\_t, [20](#)

sync\_bypass  
    hal\_adc\_sequence\_config\_t, [20](#)

trigger  
    hal\_adc\_sequence\_config\_t, [20](#)

trigger\_pol  
    hal\_adc\_sequence\_config\_t, [20](#)