Librería LPC845

Generado por Doxygen 1.8.17

Capítulo 1

Página principal de la documentacion

1.1. Introducción

Esta librería esta diseñada para ser utilizada con la línea de microcontroladores **LPC845** de la firma **NXP**. En particular, la misma actualmente está siendo desarrollada para el **LPC845** en encapsulado **QFP48** teniendo en cuenta que el mismo se encuentra en el stick de desarrollo *LPC845_BRK* .

El desarrollo está siendo orientado hacia la mayor flexibilidad posible para el usuario final, pudiendo el mismo ser un usuario con diferentes grados de conocimiento en programación orientada a sistemas embebidos así como también distintos intereses a la hora del alcance que busca en una librería.

1.2. Estructura de la librería

La librería se divide en tres capas para aportar la mayor flexibilidad posible como fue explicado en la sección Introducción. Estas tres capas están explicados a continuación.

1.2.1. Capa de aplicación de hardware (HAL)

En esta capa están definidas todas las funciones necesarias para utilizar los periféricos sin necesidad de manejar los registros del microcontrolador. Esto implica ciertas restricciones en la funcionalidad de los distintos periféricos, dadas por las implementaciones propuestas en la librería.

1.2.2. Capa de abstracción de hardware (HPL)

En esta capa están definidas todas las funciones para acceder a los distintos registros del microcontrolador de forma "humanamente legible". En caso de necesitar configuraciones particulares o no dispuestas en la *Capa de aplicación*, se deberá utilizar este nivel de abstracción. Esta capa tiene implementaciones con funcionalidades mínimas.

1.2.3. Capa de registros de hardware (HRI)

En esta capa están definidas todas las estructuras y direcciones necesarias para acceder a los distintos registros del microcontrolador en forma "directa". En caso de necesitar configuraciones o accesos no explicitados en la *Capa de abstracción de hardware* se deberá utilizar esta capa.

Capítulo 2

Indice de módulos

2.1. Módulos

Lista de todos los módulos:

Capítulo 3

Índice de estructura de datos

3.1. Estructura de datos

Lista de estructuras con una breve descripción:

Capítulo 4

Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 5

Documentación de módulos

5.1. Conversor analógico a digital (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↩ : $ADC_{res} = \frac{V_{ref_p}}{2^N}$

Esto implica que podemos preveer 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 presición 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

typedefs

- typedef void(∗ adc_sequence_interrupt_t) (void)

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_ }
- 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_A 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_SE }
- 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_INVAL }

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 **asincrónico**.*
- void hal_adc_init_sync_mode (uint32_t sample_freq, hal_adc_low_power_mode_en low_power)

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

Configuración de secuencia de *ADC*

Ejemplos

Ejemplo_ADC.c.

Campos de datos

uint16_t	channels	Canales habilitados. Cada uno de los bits representa el canal
hal_adc_trigger_sel_en	trigger	Configuración de fuente de trigger para la secuencia
hal_adc_trigger_pol_sel_en	trigger_pol	Configuración de flanco del trigger para la secuencia
hal_adc_sync_sel_en	sync_bypass	Configuración de sincronismo de la secuencia
hal_adc_interrupt_mode_en	mode	Configuración de modo de interrupcion
uint8_t	burst	Configuración de modo BURST. En caso de ser 0 esta inhabilitado, cualquier otro valor lo habilita
uint8_t	single_step	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
uint8_t	low_priority	Fijar baja prioridad de la secuencia. Unicamente 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*
adc_callback_t	callback	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*

5.1.2.2. 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

hal_adc_result_channel_en	channel	Canal que generó el resultado
uint16_t	result	Valor de la conversión

5.1.3. Documentación de los 'typedefs'

5.1.3.1. adc_sequence_interrupt_t

```
typedef void(* adc_sequence_interrupt_t) (void)
```

Tipo de dato para callback de interrupcion de sequencia

5.1.4. Documentación de las enumeraciones

5.1.4.1. 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

5.1.4.2. 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

5.1.4.3. 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

5.1.4.4. hal_adc_trigger_sel_en

enum hal_adc_trigger_sel_en

Fuente de trigger para el *ADC*

Valores de enumeraciones

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

5.1.4.5. hal_adc_trigger_pol_sel_en

enum hal_adc_trigger_pol_sel_en

Selección de polaridad del trigger del *ADC*

Valores de enumeraciones

HAL_ADC_TRIGGER_POL_SEL_NEGATIVE_EDGE	Flanco negativo
HAL_ADC_TRIGGER_POL_SEL_POSITIVE_EDGE	Flanco positivo

5.1.4.6. hal_adc_sync_sel_en

enum hal_adc_sync_sel_en

Selección de sincronismo en secuencia del *ADC*

Valores de enumeraciones

HAL_ADC_SYNC_SEL_ENABLE_SYNC	Habilitación de sincronismo
HAL_ADC_SYNC_SEL_BYPASS_SYNC	Bypass el sincronismo

5.1.4.7. 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

5.1.4.8. 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

5.1.4.9. 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

5.1.5. Documentación de las funciones

5.1.5.1. 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	*sample_freq*	Frecuencia de sampleo deseada
in	*div*	Divisor para la lógica del *ADC*
in	*clock_source*	Fuente de clock para el *ADC*
in	*low_power*	Selección de modo de bajo consumo

5.1.5.2. 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_low_power_mode_en

Parámetros

in	*sample_freq*	Frecuencia de sampleo deseada
in	*low_power*	Selección de modo de bajo consumo

Ejemplos

Ejemplo_ADC.c.

5.1.5.3. 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	*sequence*	Seleccion de secuencia a configurar
in	*config*	Configuracion deseada para la secuencia

Ejemplos

Ejemplo_ADC.c.

5.1.5.4. 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	*sequence*	Secuencia a habilitar

5.1.5.5. 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	*sequence*	Secuencia a disparar

Ejemplos

Ejemplo_ADC.c.

5.1.5.6. 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 parámetro <e>result</e>, caso contrario, guardara la cantidad de canales habilitados en la conversión en los distintos lugares del parámetro <e>result</e>.

Ver también

hal_adc_sequence_result_en

hal_adc_sequence_sel_en

hal_adc_sequence_result_t

Parámetros

in	*sequence*	Secuencia de la cual obtener el resultado
out	*result*	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_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_CTIMER.h

6.2. 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 porciento (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_CTIMER.h

6.3. 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

6.4. 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

6.5. 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

6.6. Referencia de la Estructura hal_uart_config_t

Campos de datos

- hal_uart_datalen_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

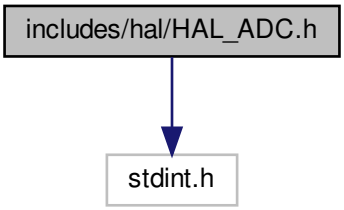
Documentación de archivos

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

typedefs

- typedef void(∗ adc_sequence_interrupt_t) (void)

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_ }
- 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_A
  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_SE }
- 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_INVAL }

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 **asincrónico**.*
- void hal_adc_init_sync_mode (uint32_t sample_freq, hal_adc_low_power_mode_en low_power)

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

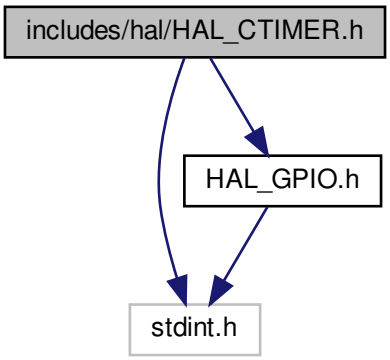
1.0

7.2. Referencia del Archivo includes/hal/HAL_CTIMER.h

Declaraciones a nivel de aplicacion del periferico CTIMER (LPC845)

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

Dependencia gráfica adjunta para HAL_CTIMER.h:



Estructuras de datos

- struct hal_ctimer_match_config_t
- struct hal_ctimer_pwm_channel_config_t
- struct hal_ctimer_pwm_config_t

Enumeraciones

- enum **hal_ctimer_match_action_en** { **HAL_CTIMER_MATCH_DO_NOGHING** = 0, **HAL_CTIMER_MA↩ TCH_CLEAR_PIN**, **HAL_CTIMER_MATCH_SET_PIN**, **HAL_CTIMER_MATCH_TOGGLE_PIN** }
- enum **hal_ctimer_match_sel_en** { **HAL_CTIMER_MATCH_0** = 0, **HAL_CTIMER_MATCH_1**, **HAL_CTI↩ MER_MATCH_2**, **HAL_CTIMER_MATCH_3** }
- enum **hal_ctimer_pwm_channel_sel_en** { **HAL_CTIMER_PWM_CHANNEL_0** = 0, **HAL_CTIMER_PW↩ M_CHANNEL_1**, **HAL_CTIMER_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	*clock_div*	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	*match_sel*	Match a configurar
in	*match_config*	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	*config*	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	*period_useg*	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

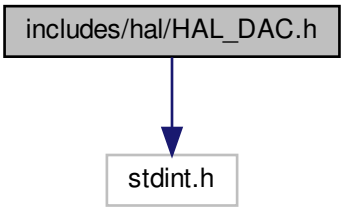
in	*channel_sel*	Seleccion de canal a configurar
in	*channel_config*	Configuracion del canal de PWM

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_SETTLIN↩ G_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	*dac*	Cual de los dos DACs inicializar
in	*settling_time*	Velocidad de conversion del DAC
in	*initial_value*	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:

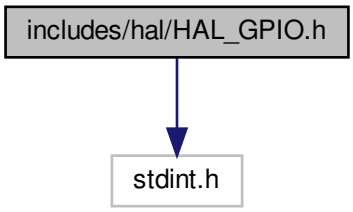
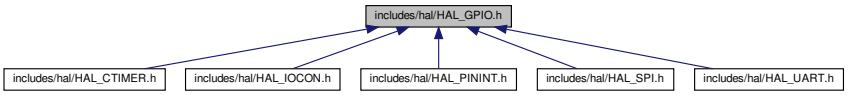


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



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_P↩ ORTPIN_0_3**,
  HAL_GPIO_PORTPIN_0_4, **HAL_GPIO_PORTPIN_0_5**, **HAL_GPIO_PORTPIN_0_6**, **HAL_GPIO_POR↩ TPIN_0_7**,
  HAL_GPIO_PORTPIN_0_8, **HAL_GPIO_PORTPIN_0_9**, **HAL_GPIO_PORTPIN_0_10**, **HAL_GPIO_PO↩ RTPIN_0_11**,
  HAL_GPIO_PORTPIN_0_12, **HAL_GPIO_PORTPIN_0_13**, **HAL_GPIO_PORTPIN_0_14**, **HAL_GPIO_P↩ ORTPIN_0_15**,
  HAL_GPIO_PORTPIN_0_16, **HAL_GPIO_PORTPIN_0_17**, **HAL_GPIO_PORTPIN_0_18**, **HAL_GPIO_P↩ ORTPIN_0_19**,
  HAL_GPIO_PORTPIN_0_20, **HAL_GPIO_PORTPIN_0_21**, **HAL_GPIO_PORTPIN_0_22**, **HAL_GPIO_P↩ ORTPIN_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	*port*	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	*portpin*	Numero de puerto/pin a configurar
in	*dir*	Direccion deseada
in	*initial_state*	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	*portpin*	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	*portpin*	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	*portpin*	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	*portpin*	Numero de puerto/pin a accionar

Devuelve

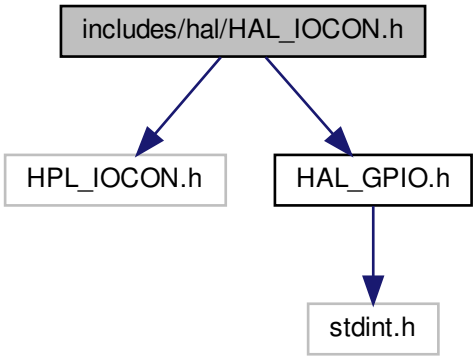
Estado actual de la GPIO

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)

  Configuracion de un pin.

7.5.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico 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

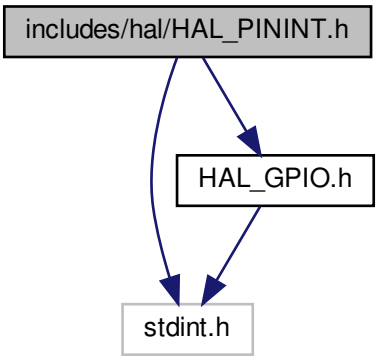
in	*portpin*	Puerto/pin a configurar
in	*pin_config*	Puntero a estructura de configuracion del pin

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	*config*	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

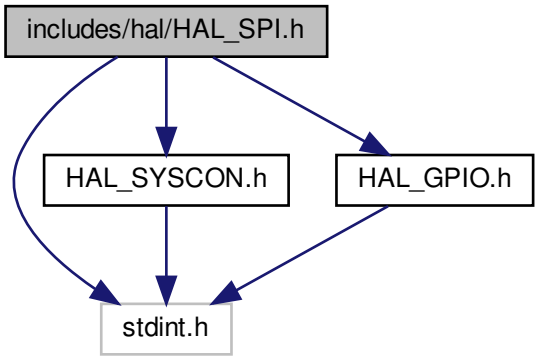
in	*channel*	Canal al cual registrar el callback
in	*new_callback*	Puntero a funcion a ejecutar

7.7. Referencia del Archivo includes/hal/HAL_SPI.h

Declaraciones a nivel de aplicacion del periferico 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

hal_spi_clock_mode_en	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	*inst*	Instancia de SPI a inicializar
in	*config*	Configuracion 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	*inst*	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	*inst*	Instancia a configurar
in	*config*	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	*inst*	Instancia a utilizar
in	*data*	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	*inst*	Instancia a configurar
in	*new_callback*	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	*inst*	Instancia a configurar
in	*new_callback*	Nuevo callback a ejecutar en RXRDY

7.8. Referencia del Archivo includes/hal/HAL_SYSCON.h

Declaraciones a nivel de aplicacion del periferico SYSCON (LPC845)

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

Dependencia gráfica adjunta para HAL_SYSCON.h:

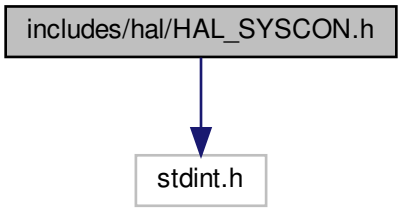
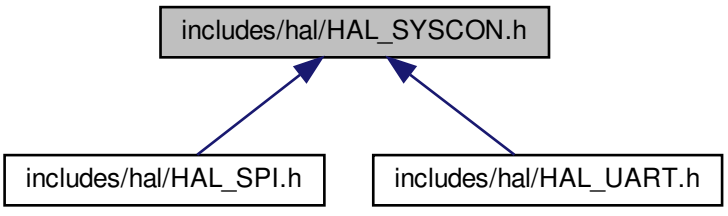


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Enumeraciones

- enum **hal_syscon_clkout_source_sel_en** { **HAL_SYSCON_CLKOUT_SOURCE_SEL_FRO** = 0, **HAL_SYSCON_CLKOUT_SOURCE_SEL_MAIN_CLOCK**, **HAL_SYSCON_CLKOUT_SOURCE_SEL_SYS_PLL**, **HAL_SYSCON_CLKOUT_SOURCE_SEL_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_FRG_CLOCK_SEL_NONE** }
- enum **hal_syscon_peripheral_sel_en** { **HAL_SYSCON_PERIPHERAL_SEL_UART0** = 0, **HAL_SYSCON_PERIPHERAL_SEL_UART1**, **HAL_SYSCON_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_SEL_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_IOCON_GLITCH_SEL_6**, **HAL_SYSCON_IOCON_GLITCH_SEL_7** }
- enum **hal_syscon_pll_source_sel_en** { **HAL_SYSCON_PLL_SOURCE_SEL_FRO** = 0, **HAL_SYSCON_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 periferico.
- uint32_t hal_syscon_get_peripheral_clock (hal_syscon_peripheral_sel_en peripheral)

  Obtener la frecuencia de clock en Hz configurada para cierto periferico.
- 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	*crystal_freq*	Frecuencia del cristal externo utilizado
in	*use_as_main*	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	*direct*	Si es distinto de cero se omite el divisor del FRO
in	*use_as_main*	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	*port*	Numero de puerto por donde sacar el clock out
in	*pin*	Numero de pin por donde sacar el clock out
in	*clock_source*	Fuente deseada para la salida clock out
in	*divider*	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	*inst*	Instancia de FRG a configurar
in	*clock_source*	Fuente de clock de entrada para el FRG
in	*mul*	Multiplicador 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 periferico.

Parámetros

in	*peripheral*	Periferico deseado
in	*clock_source*	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 periferico.

Parámetros

in	*peripheral*	Periferico deseado

Devuelve

Frecuencia en Hz del clock del periferico

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	*sel*	Seleccion de divisor
in	*div*	Valor de division 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	*clock_source*	Fuente de clock de referencia para el PLL
in	*freq*	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

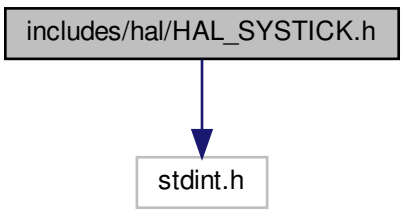
Frecuencia actual del PLL en Hz

7.9. Referencia del Archivo includes/hal/HAL_SYSTICK.h

Declaraciones a nivel de aplicacion del periferico SYSICK (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 SYSICK (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	*tick_us*	Tiempo en microsegundos deseado para el tick
in	*callback*	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

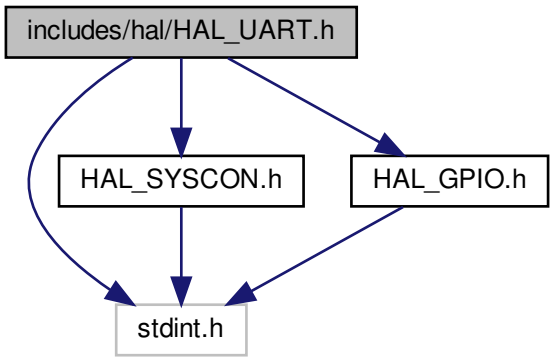
in	*callback*	Nuevo callback a ejecutar en cada tick

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_datalen_en** { **HAL_UART_DATALEN_7BIT** = 0, **HAL_UART_DATALEN_8BIT**, **HAL_U↩ ART_DATALEN_9BIT** }
- enum **hal_uart_parity_en** { **HAL_UART_PARITY_NO_PARITY** = 0, **HAL_UART_PARITY_EVEN** = 2, **H↩ AL_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	*inst*	Que instancia de UART inicializar
in	*config*	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	*inst*	Que instancia de UART usar
in	*data*	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	*inst*	Que instancia de UART usar
in	*data*	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	*inst*	A que instancia de UART registrar el callback
in	*new_callback*	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

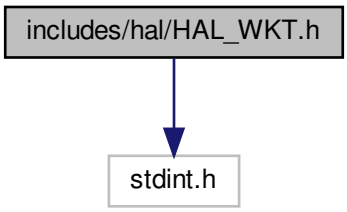
in	*inst*	A que instancia de UART registrar el callback
in	*new_callback*	Puntero a funcion a llamar cada vez que se recibe un dato por UART

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	*clock_sel*	Seleccion de clock deseada para el WKT
in	*ext_clock_value*	Valor de clock externo (si la seleccion es interna, no importa este parametro)
in	*callback*	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	*new_callback*	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.

```
/**
 * @file Ejemplo_ADC.c
 * @brief Ejemplo de utilización del \e ADC con la librería (capa de aplicación)
 *
 * El programa utiliza el clock por default con el que comienza el microcontrolador, es decir, el <em>Free
      Running
 * Oscillator</em> funcionando a \f$ 12MHz \f$.
 *
 * El periférico será configurado con las siguientes características:
 *  - Funcionamiento \b sincrónico
 *  - Frecuencia de muestreo de \f$ 1Mhz \f$
 *  - 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 \e Systick para interrumpir cada \f$ 1mseg
      \f$
 * y mediante su manejador se lleva la cuenta de los milisegundos transcurridos. Una vez transcurridos
 * \f$ 1000mseg \f$, se dispara una conversión de \e ADC, y sus resultados se guardan en dos variables
      globales.
 *
 * Ubicando un \e breakpoint adecuadamente, se pueden leer los resultados de las conversiones ya ubicadas en
 * las variables globales.
 *
 * @author Augusto Santini
 * @date 4/2020
 */
#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)
»"
»"/** Secuencia a utilizar en el ADC */
»"#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 deberia 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 ïf(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 ïf(!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;
»"}
```