Librería LPC845

Generado por Doxygen 1.8.18

Viernes, 17 de Abril de 2020 10:33:08

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 como así 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 de abstracción para aportar la mayor flexibilidad posible como fue explicado en la sección Introducción. Es recomendable que el usuario utilice la misma capa de abstracción para la utilización de la librería. En caso de no ser posible, es recomendable que al menos para cada periférico se utilice una única capa de abstracción. Estas tres capas están explicados a continuación.

Nota

Toda la documentación acerca de los distintos periféricos implementados en la documentación de la librería, están referidos a la capa Capa de aplicación de hardware (HAL), pero se explica la estructura de la librería dado que usuarios más avanzados en el campo de sistemas embebidos puedan llegar a utilizar con mayor frecuencia las dos capas inferiores.

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.

1.3. Acerca del stick de desarrollo LPC845_BRK

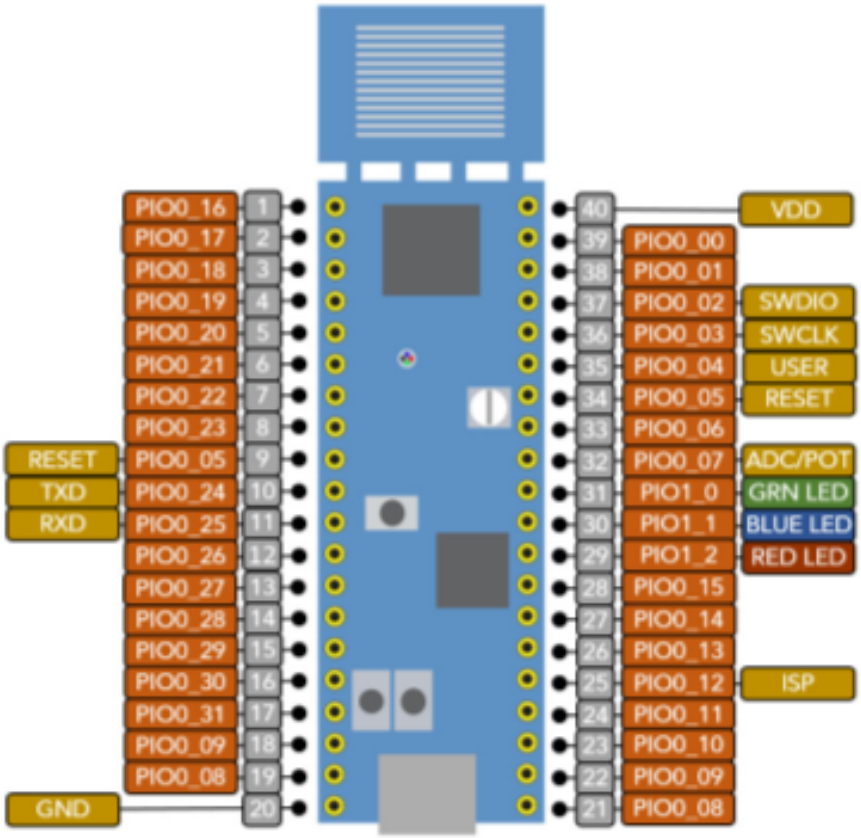


Figura 1.1 Pinout del LPC845_BRK

El stick de desarrollo utilizado a lo largo del desarrollo de esta librería, también utilizado para la demostración de todos los ejemplos adjuntos, es el indicado en la imágen superior. El mismo viene dotado de varios componentes útiles a la hora de desarrollar un proyecto, evitándonos en gran medida realizar nuestro propio hardware, ahorrándonos así tiempo de desarrollo causado por errores en el desarrollo del hardware, fabricación de prototipos y demás. Dichos componentes son:

- LED RGB: LEDs *rojo*, *verde* y *azul* en un mismo sustrato. Los tres LEDs se encuentran en paralelo, por lo que **no es posible encender más de un LED a la vez**. Estos LEDs permiten pruebas de salidas con el periférico Entradas/Salidas de Propósito General (GPIO). Los puertos/pines de los mismos son:
  - Rojo: Puerto 1 ; Pin 2
  - Azul: Puerto 1 ; Pin 1
  - Verde: Puerto 1 ; Pin 0

- Preset de una vuelta sin tope: Resistor variable de tres terminales, muy útil para pruebas de Conversor analógico a digital (ADC). Ubicado en:
  - Puerto 0 ; Pin 7
- Tres pulsadores:
  - Pulsador de usuario: Útil para pruebas relacionadas con lectura de Entradas/Salidas de Propósito General (GPIO) interrupciones de pin Interrupciones de pin / Motor de patrones (PININT). El pulsador de usuario también se encuentra en un pin disponible para despertar al microcontrolador de modos de funcionamiento de bajo consumo. Ubicado en: Puerto 0 ; Pin 4
  - Pulsador de reset: Con este pulsador se puede generar un *RESET* por hadrware en caso de habilitar dicha función en el pin correspondiente. En caso de no habilitar dicha función, se puede utilizar el pulsador como un pulsador normal. Ubicado en: Puerto 0 ; Pin 5
  - Pulsador para modo ISP: La principal utilidad de este pulsador, es en conjunto con el de reset, para entrar a modo de programación ISP, para recuperar el microcontrolador de ciertos estados de falla.
- Conexión con un emulador de puerto serie: El chip que hace las veces de programador en el stick de desarrollo, cuenta con el software necesario para emular un puerto serie en un puerto USB. Esto es una gran ventaja, dado que no necesitamos ninguna interfaz adicional al stick de desarrollo para realizar pruebas con el periférico UART. Ubicación de pines en:
  - Pin RX (Receptor del emulador): Puerto 0 ; Pin 25
  - Pin TX (Transmisor del emulador): Puerto 0 ; Pin 24
- Pad táctil: El stick de desarrollo tiene en su borde opuesto al puerto USB, un pad táctil. El mismo puede ser utilizado por el microcontrolador y el periférico *CAPTOUCH*.

Cabe destacar que ciertos pines no estén disponibles o tengan componentes conectados, por lo cual es altamente recomendable revisar las conexiones en cada pin a utilizar, en el esquemático del stick de desarrollo.

1.4. Utilización de la librería en proyectos MCUXpresso

Existen básicamente dos formas de utilizar la librería.

1.4.0.1. Compilación de librería en proyecto externo

En ciertos casos es deseable en el proyecto a trabajar, que la librería esté compilada previamente en un proyecto separado, para tener mayor control entre los distintos proyectos que utilicen la librería, y para un mayor encapsulamiento de las funciones. Con esta estrategia, cualquier cambio realizado en la librería, afectará a todos los proyectos que la utilicen. Para utilizar esta estrategia, seguir los siguientes pasos:

1. Compilar la librería en un proyecto de tipo **Librería estática**
2. En el proyecto que desee utilizar la librería, configurar las siguientes propiedades bajo las propiedades C/C++ Build -> Settings:
   - *MCU C Compiler -> Includes -> Include paths*: Indicar el directorio a la capa de abstracción a utilizar del proyecto de librería estática ya compilado correctamente. Ejemplo: "${workspace_loc:/Libreria_L↩ PC845/includes/hal}"
   - *MCU Linker -> Libraries -> Libraries*: Indicar el nombre de la librería compilada. Ejemplo: "Libreria_↩ LPC845"
   - *MCU Linker -> Libraries -> Library search path*: Indicar el directorio donde la librería estática haya sido compilada. Dicho directorio dependerá de si la compilación de la librería fue hecha en modo *Release* o *Debug*. Ejemplo: "${workspace_loc:/Libreria_LPC845/Debug}"
3. En este punto debería poder utilizar la librería sin ningún problema

Las opciones de compilación de la librería como pueden ser niveles de optimización, quedan en este caso a cargo del usuario y son modificables en las configuraciones del proyecto de la librería.

1.4.0.2. Agregado de archivos fuentes de librería necesarios

Si es necesario agregar/quitar funcionalidades de la librería, o se desea cambiar implementaciones ya realizadas, solo es necesario agregar los archivos de cabecera y fuentes necesarios al proyecto donde se va a trabajar. Esta opción permite una personalización de la librería, así como utilización de menos espacio de memoria de código, dado que el compilador probablemente descarte todas las funciones que no sean utilizadas.

Capítulo 2

Documentación de módulos

2.1. Comparador analógico (ACMP)

2.1.1. Descripción detallada

Introducción

Este periférico compara 2 señales analógicas y responde con una salida digital que indica cuál de ellas es la mayor. Es posible conectar la señal de salida del comparador a un pin digital de salida.

El comparador tiene 2 entradas, una considerada positiva y otra negativa.

- Si la señal conectada a la entrada positiva es mayor, la salida del comparador estará en estado alto.
- Si la señal conectada a la entrada negativa es mayor, la salida del comparador estará en estado bajo.

Este periférico cuenta además con la posibilidad de generar un pedido de interrupción según el tipo de flanco generado por un cambio a la salida del comparador.

Selección de las entradas analógicas al comparador

Para este microcontrolador ambas entradas del comparador son seleccionables entre entradas analógicas externas y algunas señales internas del microcontrolador.

Las entradas seleccionables son:

- Tensión de salida de la *Voltage Ladder*.
- 5 entradas analógicas externas.
- Tensión de referencia interna *Bandgap*.
- Salida del conversor digital a analógico *DAC*.

Voltage Ladder

La *Voltage Ladder* puede ser utilizada para generar una tensión determinada a partir de una tensión externa, o de la propia alimentación del microcontrolador V_{DD}.

La tensión de salida es configurable por el usuario y responde a la siguiente ecuación (en función de la tensión de referencia elegida):

$$V_{LAD} = \frac{n * V_{ref}}{31}$$

Donde:

- n: Entero entre 0 y 31.
- V_{ref}: Tensión de referencia de la *Voltage Ladder*.

Salida del comparador como trigger de hardware de otros periféricos

Es posible utilizar la salida del comparador como trigger por hardware para otros periféricos para que estos lleven a cabo alguna funcionalidad sin la necesidad de intervención por software del microcontrolador .

De hecho, este es uno de los triggers por hardware de disparo de conversiones del periférico *ADC*.

Campos de aplicación típicos

Loren Ipsum

Estructuras de datos

- struct hal_acpm_config_t
- struct hal_acmp_ladder_config_t

Enumeraciones

- enum hal_acmp_output_control_en {
  HAL_ACMP_OUTPUT_DIRECT = 0,
  HAL_ACMP_OUTPUT_SYNC }
- enum hal_acmp_hysteresis_sel_en {
  HAL_ACMP_HYSTERESIS_NONE = 0,
  HAL_ACMP_HYSTERESIS_5mV,
  HAL_ACMP_HYSTERESIS_10mV,
  HAL_ACMP_HYSTERESIS_20mV }
- enum hal_acmp_ladder_vref_sel_en {
  HAL_ACMP_LADDER_VREF_VDD = 0,
  HAL_ACMP_LADDER_VREF_VDDCMP_PIN }
- enum hal_acmp_edge_sel_en {
  HAL_ACMP_EDGE_FALLING = 0,
  HAL_ACMP_EDGE_RISING,
  HAL_ACMP_EDGE_BOTH }
- enum hal_acmp_input_voltage_sel_en {
  HAL_ACMP_INPUT_VOLTAGE_VLADDER_OUT = 0,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I1,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I2,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I3,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I4,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I5,
  HAL_ACMP_INPUT_VOLTAGE_BANDGAP,
  HAL_ACMP_INPUT_VOLTAGE_DACOUT0 }

Funciones

- void hal_acmp_init (void)

  Inicialización del periférico Comparador Analógico.
- void hal_acmp_deinit (void)

  De-inicialización del periferico Comparador Analógico.
- void hal_acmp_config (const hal_acpm_config_t ∗acmp_config)

  Configuración de parámetros generales del comparador analógico.
- void hal_acmp_ladder_config (const hal_acmp_ladder_config_t ∗config)

  Configuración de la Voltage Ladder del comparador analógico.
- void hal_acmp_input_select (hal_acmp_input_voltage_sel_en positive_input, hal_acmp_input_voltage_sel_en negative_input)

  Selecciona las entradas positiva y negativa deseadas para el comparador.
- void hal_acmp_output_pin_set (hal_gpio_portpin_en port_pin)

  Asigna la salida del comparador a un pin externo del microcontrolador.
- void hal_acmp_output_pin_clear ()

  Deshace la asignación de la salida del comparador a un pin externo del microcontrolador.

2.1.2. Documentación de las estructuras de datos

2.1.2.1. struct hal_acpm_config_t

Estructura de configuración de características generales del comparador analógico.

Campos de datos

hal_acmp_output_control_en	output_control	Control de sincronismo de la señal de salida del comparador.
hal_acmp_hysteresis_sel_en	hysteresis	Configura umbral de histérisis para reflejar cambios en la salida del comparador.
uint8_t	interrupt_enable	Habilita/deshabilita interrupciones generadas por la salida del comparador.
hal_acmp_edge_sel_en	edge_sel	Configura el tipo de flanco de la señal de salida del comparador que genera un pedido de interrupción.

2.1.2.2. struct hal_acmp_ladder_config_t

Estructura de configuración de la Voltage Ladder del comparador analógico.

Campos de datos

uint8_t	enable	Habilitación/deshabilitación de la Voltage Ladder.
hal_acmp_ladder_vref_sel_en	vref_sel	Selección de tensión de referencia de la Voltage Ladder.
uint8_t	step	Configura qué fracción de la tensión de referencia estará será la tensión de salida de la Voltage Ladder.

2.1.3. Documentación de las enumeraciones

2.1.3.1. hal_acmp_output_control_en

`enum hal_acmp_output_control_en`

Selección de sincronismo de la señal digital de salida del comparador analógico.

Valores de enumeraciones

HAL_ACMP_OUTPUT_DIRECT	Los cambios se reflejarán instantáneamente.
HAL_ACMP_OUTPUT_SYNC	Los cambios serán sincronizados con el clock de bus.

2.1.3.2. hal_acmp_hysteresis_sel_en

`enum hal_acmp_hysteresis_sel_en`

Posible umbral de histéresis para cambios a la salida del comparador analógico.

Valores de enumeraciones

HAL_ACMP_HYSTERESIS_NONE	Sin histéresis.
HAL_ACMP_HYSTERESIS_5mV	Umbral de 5mV.
HAL_ACMP_HYSTERESIS_10mV	Umbral de 10mV.
HAL_ACMP_HYSTERESIS_20mV	Umbral de 20mV.

2.1.3.3. hal_acmp_ladder_vref_sel_en

`enum hal_acmp_ladder_vref_sel_en`

Posibles tensiones de referencia para la Voltage Ladder del comparador analógico.

Valores de enumeraciones

HAL_ACMP_LADDER_VREF_VDD	Tensión de referencia VDD.
HAL_ACMP_LADDER_VREF_VDDCMP_PIN	Tensión de referencia tomada del pin externo con función analógica VDDCMP.

2.1.3.4. hal_acmp_edge_sel_en

```
enum hal_acmp_edge_sel_en
```

Selección de tipo de flanco de la señal de salida del comparador analógico que será trigger de interrupción.

Valores de enumeraciones

HAL_ACMP_EDGE_FALLING	Flanco descendente.
HAL_ACMP_EDGE_RISING	Flanco ascendente.
HAL_ACMP_EDGE_BOTH	Ambos tipos de flanco.

2.1.3.5. hal_acmp_input_voltage_sel_en

```
enum hal_acmp_input_voltage_sel_en
```

Posibles entradas al comparador analógico, válidas tanto para la positiva como la negativa.

Valores de enumeraciones

HAL_ACMP_INPUT_VOLTAGE_VLADDER_OUT	Tensión interna de salida de la Voltage Ladder.
HAL_ACMP_INPUT_VOLTAGE_ACMP_I1	Pin externo con función analógica ACMP_I1.
HAL_ACMP_INPUT_VOLTAGE_ACMP_I2	Pin externo con función analógica ACMP_I2.
HAL_ACMP_INPUT_VOLTAGE_ACMP_I3	Pin externo con función analógica ACMP_I3.
HAL_ACMP_INPUT_VOLTAGE_ACMP_I4	Pin externo con función analógica ACMP_I4.
HAL_ACMP_INPUT_VOLTAGE_ACMP_I5	Pin externo con función analógica ACMP_I5.
HAL_ACMP_INPUT_VOLTAGE_BANDGAP	Tensión interna de referencia 'Bandgap'.
HAL_ACMP_INPUT_VOLTAGE_DACOUT0	Señal interna, salida 0 del periférico DAC.

2.1.4. Documentación de las funciones

2.1.4.1. hal_acmp_init()

```
void hal_acmp_init (
            void  )
```

Inicialización del periférico Comparador Analógico.

Ver también

hal_acmp_deinit

hal_acmp_config

2.1.4.2. hal_acmp_deinit()

```
void hal_acmp_deinit (
            void  )
```

De-inicialización del periferico Comparador Analógico.

Además de la usual deinicialización de un periférico, libera todos los pines posiblemente utilizados por el comparador de su función analógica.

2.1.4.3. hal_acmp_config()

```
void hal_acmp_config (
            const hal_acpm_config_t * acmp_config )
```

Configuración de parámetros generales del comparador analógico.

Esta función configura las siguientes características del comparador:

- Sincronización de la señal de salida del comparador con clock de bus.
- Rango de histérisis sobre la comparación para validar un cambio de la salida del comparador.
- Habilitación o deshabilitación de interrupciones, y según qué tipo de flanco se quiera de la salida del comparador.

Parámetros

in	*acmp_config*	Puntero a estructura con parámetros a configurar.

Ver también

hal_acpm_config_t
hal_acmp_ladder_config
hal_acmp_input_select

2.1.4.4. hal_acmp_ladder_config()

```
void hal_acmp_ladder_config (
            const hal_acmp_ladder_config_t * ladder_config )
```

Configuración de la Voltage Ladder del comparador analógico.

Configura las siguientes características de la Voltage Ladder del comparador:

- Habilitación o no.
- Tensión de referencia.
- Fracción ('step') utilizada de dicha tensión de referencia.

Parámetros

in	*config*	Puntero a estructura con parámetros de configuración deseados de la Voltage Ladder.

Ver también

hal_acmp_ladder_config_t
hal_acmp_config
hal_acmp_ladder_config
hal_acmp_input_select
hal_acmp_deinit

Configuración de la Voltage Ladder del comparador analógico.

Configura las siguientes características de la escalera de tensión del comparador:

- Habilitación o no.
- Tensión de referencia.
- Fracción ('step') utilizada de dicha tensión de referencia.

Parámetros

in	*config*	Puntero a estructura con parámetros de configuración deseados de la escalera de tensión.

Ver también

hal_acmp_ladder_config_t
hal_acmp_config
hal_acmp_ladder_config
hal_acmp_input_select
hal_acmp_deinit

2.1.4.5. hal_acmp_input_select()

```
void hal_acmp_input_select (
            hal_acmp_input_voltage_sel_en positive_input,
            hal_acmp_input_voltage_sel_en negative_input )
```

Selecciona las entradas positiva y negativa deseadas para el comparador.

Para entradas analógicas además realiza la configuración necesaria del pin externo.

Parámetros

in	*positive_input*	Selección de entrada para la entrada positiva del comparador analógico.
in	*negative_input*	Selección de entrada para la entrada negativa del comparador analógico.

Ver también

hal_acmp_input_voltage_sel_en

hal_acmp_ladder_config

hal_acmp_deinit

Para entradas analógicas además realiza la configuración necesaria del pin externo de entrada analógica.

Parámetros

in	*positive_input*	Selección de entrada para la entrada positiva del comparador analógico.
in	*negative_input*	Selección de entrada para la entrada negativa del comparador analógico.

Ver también

hal_acmp_input_voltage_sel_en

hal_acmp_ladder_config

hal_acmp_deinit

2.1.4.6. hal_acmp_output_pin_set()

```
void hal_acmp_output_pin_set (
            hal_gpio_portpin_en port_pin )
```

Asigna la salida del comparador a un pin externo del microcontrolador.

Nota

: No realiza ningún tipo de configuración del pin respecto a sus resistores.

Parámetros

in	*port_pin*	Indica puerto y pin deseado.

Ver también

hal_acmp_output_pin_clear

2.1.4.7. hal_acmp_output_pin_clear()

```
void hal_acmp_output_pin_clear ( )
```

Deshace la asignación de la salida del comparador a un pin externo del microcontrolador.

Ver también

hal_acmp_output_pin_set

2.2. Conversor analógico a digital (ADC)

2.2.1. Descripción detallada

Introducción

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:

$$ADC_{res} = \frac{V_{ref_p}}{2^N}$$

Siendo N la cantidad de bits disponibles en la conversión. Esto implica que podemos preveer el valor resultante de la conversión analógica/digital mediante la siguiente ecuación:

$$ADC_{conv} = \frac{V_{ADC_{in}}}{ADC_{res}}$$

Nota

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 habilitados en dicha secuencia. Se tienen dos secuencias configurables (*Secuencia A y Secuencia B*), las cuales se pueden configurar de forma tal que un disparo de *Secuencia B* interrumpa a una conversión actual de la *Secuencia A*.

Nota

Las secuencias de conversión son configuradas mediante la función hal_adc_sequence_config.

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 la escritura de un registro de control del *ADC*. El software puede tener total control sobre el punto de inicio de conversión.
2. Iniciadas por hardware: Las secuencias de conversión son iniciadas dependiendo de otras señales, sean las mismas internas o externas al microcontrolador.

Nota

En caso de disparar conversiones por software, se utiliza la función hal_adc_sequence_start para dicho propósito. En caso de que las conversiones sean iniciadas por hardware, no se debe llamar a ninguna función, y la secuencia de conversión se disprará cuando suceda el evento configurado en la secuencia.

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 las funciones hal_adc_init_sync_mode y hal_adc_init_async_mode.

Nota

La autocalibración debe realizarse cuando el **microcontrolador** sale de un modo de funcionamiento de bajo consumo, no cuando el periférico *ADC* sale de modo bajo consumo.

Modo sicnrónico/asincrónico

En este microcontrolador, el *ADC* puede ser configurado de dos formas distintas en cuanto a sus clocks:

- Modo asincrónico: El clock que alimenta a la lógica de muestreo del periférico puede ser de una naturaleza distinta al clock del sistema que alimenta a la lógica del periférico.
- Modo sincrónico: El clock que alimenta tanto a la lógica de muestreo del periférico como la lógica del periférico, estan en sincronismo.

Nota

La configuración de esta característica se realiza en la función hal_adc_init_sync_mode o hal_adc_init_async_mode dependiendo de las necesidades del usuario.

Modo bajo consumo

El periférico dispone de una funcionalidad configurable de bajo consumo. Si la misma está habilitada, en cualquier momento que el *ADC* no esté realizando alguna conversión, la energía del mismo se reducirá, permitiendo así tener un menor consumo. El costo de este modo de funcionamiento, es un delay extra cada vez que se dispara una nueva conversión, dado que el periférico deberá salir del modo bajo consumo. Consultar el manual de usuario del microcontrolador para más información.

Nota

El parámetro de bajo consumo se configura en las funciones de inicialización hal_adc_init_sync_mode o hal_adc_init_async_mode.

Velocidad de conversión/Frecuencia de muestreo

Cada conversión realizada toma un tiempo que dependerá del clock configurado en el periférico. Podemos obtener este tiempo de conversión mediante la ecuación:

$$t_{conv_{ADC}} = \frac{1}{f_{ADC}/25}$$

$$f_{muestreo_{ADC}} = \frac{1}{t_{conv_{ADC}}}$$

La división por 25 en el denominador, es debido a la naturaleza del periférico de *aproximaciones sucesivas*. 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 frecuencia de 25MHz obtenemos el tiempo tomado por cada conversión:

$$t_{conv_{ADC}} = \frac{1}{25MHz/25}$$
$$t_{conv_{ADC}} = 1\mu s$$
$$f_{muestreo_{ADC}} = 1MHz$$

Esto implica que entre un inicio de conversión y la finalización de la misma, pasará 1 microsegundo. Nótese que este tiempo corresponde a una conversión para un único canal. En caso de estar convirtiendo varios canales, se deberá multiplicar dicho tiempo de conversión 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 (asumiendo que se dispara una conversión de secuencia completa).

El *ADC* no puede convertir a cualquier frecuencia de muestreo, existen frecuencias máximas dependiendo del tipo de funcionamiento configurado para el periférico:

- Funcionamiento en modo *sincrónico*: Frecuencia de muestreo máxima de 1.2MHz
- Funcionamiento en modo *asincrónico*: Frecuencia de muestreo máxima de 0.6MHz

Nota

La frecuencia de muestreo se configura en las funciones de inicialización hal_adc_init_sync_mode o hal_adc_init_async_mode.

Campos de aplicación típicos

- Audio/Video
- Señales de naturaleza biológica (ECG, EEG)
- Entradas de usuario de hardware (Preset, Potenciómetro)
- Sensores con salida analógica de variables físicas (Termómetro, Luxómetro)

Consideraciones acerca de los callbacks

Los callbacks asociados en las configuraciones posibles son ejecutados en el contexto de una **interrupción**, por lo que el usuario deberá tener las consideraciones adecuadas a la hora de lo que realiza el callback asociado. Ver adc_sequence_interrupt_t y adc_comparison_interrupt_t.

Estructuras de datos

- struct hal_adc_sequence_config_t
- struct hal_adc_sequence_result_t
- struct hal_adc_channel_compare_result_t

typedefs

- typedef void(∗ adc_sequence_interrupt_t) (void)

  Tipo de dato para callback de interrupcion de sequencia.
- typedef void(∗ adc_comparison_interrupt_t) (void)

  Tipo de dato para callback de interrupcion de comparación.

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 }
- enum hal_adc_threshold_sel_en {
  HAL_ADC_THRESHOLD_SEL_0 = 0,
  HAL_ADC_THRESHOLD_SEL_1 }

- enum hal_adc_threshold_interrupt_sel_en {
  HAL_ADC_THRESHOLD_IRQ_SEL_DISABLED = 0,
  HAL_ADC_THRESHOLD_IRQ_SEL_OUTSIDE,
  HAL_ADC_THRESHOLD_IRQ_SEL_CROSSING }
- enum hal_adc_compare_range_result_en {
  HAL_ADC_COMPARISON_RANGE_INSIDE = 0,
  HAL_ADC_COMPARISON_RANGE_BELOW,
  HAL_ADC_COMPARISON_RANGE_ABOVE }
- enum hal_adc_compare_crossing_result_en {
  HAL_ADC_COMPARISON_NO_CROSSING = 0,
  HAL_ADC_COMPARISON_CROSSING_DOWNWARD = 2,
  HAL_ADC_COMPARISON_CROSSING_UPWARD }

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_sequence_config (hal_adc_sequence_sel_en sequence, const hal_adc_sequence_config_t *config)

  Configurar una secuencia de conversión.
- void hal_adc_sequence_start (hal_adc_sequence_sel_en sequence)

  Disparar conversiones en una secuencia.
- void hal_adc_sequence_stop (hal_adc_sequence_sel_en sequence)

  Detener conversiones en una secuencia de conversión.
- hal_adc_sequence_result_en hal_adc_sequence_get_result (hal_adc_sequence_sel_en sequence, hal_adc_sequence_result_t *result)

  Obtener resultado de la secuencia.
- void hal_adc_threshold_config (hal_adc_threshold_sel_en threshold, uint16_t low, uint16_t high)

  Configurar valor de umbral de comparación.
- void hal_adc_threshold_channel_config (uint8_t adc_channel, hal_adc_threshold_sel_en threshold, hal_adc_threshold_interrupt_sel_en irq_mode)

  Configura un canal para utilizar la funcionalidad de comparación con un umbral y su tipo de interrupción deseada.
- void hal_adc_threshold_register_interrupt (adc_comparison_interrupt_t callback)

  Registrar un callabck de interrupción para interrupción por threshold.
- void hal_adc_threshold_get_comparison_results (hal_adc_channel_compare_result_t *results)

  Obtener resultados de comparación de la última conversión.

2.2.2. Documentación de las estructuras de datos

2.2.2.1. 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_sequence_interrupt_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*

2.2.2.2. 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

2.2.2.3. struct hal_adc_channel_compare_result_t

Resultado de comparaciones

Ejemplos

Ejemplo_ADC.c.

Campos de datos

hal_adc_result_channel_en	channel	Canal que detectó comparación configurada
uint16_t	value	Valor de la conversión
hal_adc_compare_range_result_en	result_range	Rango de la comparación
hal_adc_compare_crossing_result_en	result_crossing	Resultado si está cruzando

2.2.3. Documentación de los 'typedefs'

2.2.3.1. adc_sequence_interrupt_t

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

Tipo de dato para callback de interrupcion de sequencia.

Nota

Estos callbacks son ejecutados desde un contexto de interrupción, por lo que el usuario deberá tener todas las consideraciones necesarias al respecto.

2.2.3.2. adc_comparison_interrupt_t

```
typedef void(* adc_comparison_interrupt_t) (void)
```

Tipo de dato para callback de interrupcion de comparación.

Nota

Estos callbacks son ejecutados desde un contexto de interrupción, por lo que el usuario deberá tener todas las consideraciones necesarias al respecto.

2.2.4. Documentación de las enumeraciones

2.2.4.1. 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

2.2.4.2. 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

2.2.4.3. 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

2.2.4.4. 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

2.2.4.5. 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

2.2.4.6. 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

2.2.4.7. 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

2.2.4.8. 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

2.2.4.9. 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

2.2.4.10. hal_adc_threshold_sel_en

`enum hal_adc_threshold_sel_en`

Selección del umbral del ADC.

Valores de enumeraciones

HAL_ADC_THRESHOLD_SEL↩ _0	Banco 0 de threshold
HAL_ADC_THRESHOLD_SEL↩ _1	Banco 1 de threshold

2.2.4.11. hal_adc_threshold_interrupt_sel_en

`enum hal_adc_threshold_interrupt_sel_en`

Posibles configuraciones de la interrupción por comparación de las muestras obtenidas de un canal con el umbral establecido.

Valores de enumeraciones

HAL_ADC_THRESHOLD_IRQ_SEL_DISABLED	Interrupción por umbral inhabilitada
HAL_ADC_THRESHOLD_IRQ_SEL_OUTSIDE	Interrupción por conversión fuera del umbral establecido
HAL_ADC_THRESHOLD_IRQ_SEL_CROSSING	Interrupción por conversión cruzando alguno de los umbrales establecidos

2.2.4.12. hal_adc_compare_range_result_en

`enum hal_adc_compare_range_result_en`

Resultado de comparación de conversión contra los umbrales

Valores de enumeraciones

HAL_ADC_COMPARISON_RANGE_INSIDE	Resultado de conversión dentro del umbral estipulado
HAL_ADC_COMPARISON_RANGE_BELOW	Resultado de conversión por debajo del umbral estipulado
HAL_ADC_COMPARISON_RANGE_ABOVE	Resultado de conversión por encima del umbral estipulado

2.2.4.13. hal_adc_compare_crossing_result_en

`enum hal_adc_compare_crossing_result_en`

Resultado de comparación de conversión (cruce)

Valores de enumeraciones

HAL_ADC_COMPARISON_NO_CROSSING	Resultado de conversión no estaba cruzando ningún umbral
HAL_ADC_COMPARISON_CROSSING_DOWNWARD	El resultado de la conversión actual cruzó algún umbral hacia abajo
HAL_ADC_COMPARISON_CROSSING_UPWARD	El resultado de la conversión actual cruzó algún umbral hacia arriba

2.2.5. Documentación de las funciones

2.2.5.1. 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

Precondición

Configuración del clock a utilizar

2.2.5.2. 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.

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	*low_power*	Selección de modo de bajo consumo

Precondición

Configuración del clock a utilizar

Ejemplos

Ejemplo_ADC.c.

2.2.5.3. hal_adc_deinit()

```
void hal_adc_deinit (
            void  )
```

De-inicialización del *ADC*.

Además, desliga todos los pines externos posiblemente utilizados por el ADC de su función analógica.

2.2.5.4. hal_adc_sequence_config()

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

Configurar una secuencia de conversión.

Una vez configurado un canal del ADC con este método, el pin externo correspondiente a él quedará ligado a su función analógica hasta de-inicializar el periférico por medio de la función hal_adc_deinit()

Nota

Esta función no habilita las secuencias.

Parámetros

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

Ver también

hal_adc_sequence_sel_en

hal_adc_sequence_config_t

Ejemplos

Ejemplo_ADC.c.

2.2.5.5. hal_adc_sequence_start()

```
void hal_adc_sequence_start (
            hal_adc_sequence_sel_en sequence )
```

Disparar conversiones en una secuencia.

Precondición

Si se están utilizando triggers por hardware, esta función simplemente habilitará la secuencia, sin disparar una conversión. Sin embargo, para evitar un trigger espúrio, es necesario asegurar que la señal de trigger se encuentre inactiva según cómo se lo haya definido en el parámetro hal_adc_sequence_config_t::trigger_pol

Si la secuencia está configurada en modo hal_adc_sequence_config_t::burst esta función comenzará conversiones consecutivas en todos los canales configurados.

Si la secuencia **no** utiliza el modo hal_adc_sequence_config_t::burst, entonces esta función disparará una sola conversión en un canal o una conversión en todos los canales configurados de la secuencia, dependiendo del parámetro hal_adc_sequence_config_t::single_step.

Nota

En todos los casos, esta función habilita la secuencia antes de disparar la conversión (si corresponde).

Parámetros

in	*sequence*	Secuencia a disparar

Ver también

hal_adc_sequence_sel_en

hal_adc_sequence_start

hal_adc_sequence_config_t

Ejemplos

Ejemplo_ADC.c.

2.2.5.6. hal_adc_sequence_stop()

```
void hal_adc_sequence_stop (
            hal_adc_sequence_sel_en sequence )
```

Detener conversiones en una secuencia de conversión.

Si hay una conversión en curso, al finalizar ésta ya no se realizará otra.

Nota

Esta función, además, deshabilita la secuencia. Esto es necesario para cambiar las configuraciones de secuencias ya en uso de forma segura.

Parámetros

in	*sequence*	Secuencia a detener

Ver también

hal_adc_sequence_sel_en

hal_adc_start_sequence

2.2.5.7. hal_adc_sequence_get_result()

```
hal_adc_sequence_result_en hal_adc_sequence_get_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 hal_adc_sequence_config_t::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 hal_adc_sequence_result_t::result, caso contrario, guardara la cantidad de canales habilitados en la conversión en los distintos lugares del parámetro *result*.

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

El usuario debe garantizar que existe lugar para la misma cantidad de canales habilitados en la secuencia.

Devuelve

Resultado de la función

Ejemplos

Ejemplo_ADC.c.

2.2.5.8. hal_adc_threshold_config()

```
void hal_adc_threshold_config (
            hal_adc_threshold_sel_en threshold,
            uint16_t low,
            uint16_t high )
```

Configurar valor de umbral de comparación.

Parámetros

in	*threshold*	Selección de umbral a configurar
in	*low*	Umbral bajo
in	*high*	Umbral alto

Ver también

hal_adc_threshold_sel_en

Ejemplos

Ejemplo_ADC.c.

2.2.5.9. hal_adc_threshold_channel_config()

```
void hal_adc_threshold_channel_config (
            uint8_t adc_channel,
            hal_adc_threshold_sel_en threshold,
            hal_adc_threshold_interrupt_sel_en irq_mode )
```

Configura un canal para utilizar la funcionalidad de comparación con un umbral y su tipo de interrupción deseada.

Puede utilizarse como método de deshabilitación de las interrupciones, debidas a comparación, de canales específicos. Para deshabilitar las interrupciones, debidas a comparación, de todos los canales a la vez puede utilizarse la función hal_adc_threshold_register_interrupt()

Parámetros

in	*adc_channel*	Canal de ADC en el cual configurar el umbral
in	*threshold*	Selección de umbral a configurar
in	*irq_mode*	Indica el tipo evento por el cual la comparación con el umbral dispara la interrupción, o la desactiva.

Ver también

hal_adc_threshold_sel_en

hal_adc_threshold_interrupt_sel_en

Ejemplos

Ejemplo_ADC.c.

2.2.5.10. hal_adc_threshold_register_interrupt()

```
void hal_adc_threshold_register_interrupt (
            void(*)(void) callback )
```

Registrar un callabck de interrupción para interrupción por threshold.

Si se le pasa NULL, esta función efectivamente deshabilita las interrupciones, debidas a comparaciones, de todos los canales del ADC. Sin embargo, esto no altera en modo alguno la configuración ya establecida por el usuario para las comparaciones de cada canal.

En caso de querer volver a habilitar las interrupciones debidas a comparación, de todos los canales del ADC configurados para ello de antemano, con simplemente llamar a esta función con un puntero a función válido (no NULL) para la interrupción es suficiente.

Parámetros

in	*callback*	Callback a ejecutar en interrupción por threshold

Ver también

hal_adc_threshold_channel_config

Ejemplos

Ejemplo_ADC.c.

2.2.5.11. hal_adc_threshold_get_comparison_results()

```
void hal_adc_threshold_get_comparison_results (
            hal_adc_channel_compare_result_t * results )
```

Obtener resultados de comparación de la última conversión.

Parámetros

out	*results*	Puntero a donde guardar los resultados

El usuario **debe** garantizar que hayan por lo menos la cantidad de memoria reservada de este tipo como cantidad de canales habilitados para comparar contra un umbral.

Ver también

hal_adc_channel_compare_result_t

Ejemplos

Ejemplo_ADC.c.

2.3. Conversor digital a analógico (DAC)

2.3.1. Descripción detallada

Introducción

El *DAC*, como su nombre lo indica, convierte un valor digital en una tensión analógica. En el caso de este microcontrolador, tiene dos módulos *DACs* independientes con una resolución de 10 bits cada uno. Se puede calcular la resolución de cada paso de *DAC* como:

$$DAC_{res} = \frac{V_{ref_p}}{2^N}$$

Siendo N la cantidad de bits de resolución del *DAC*. Esto implica que podemos preveer el valor de salida de tensión analógica conociendo el valor a convertir por el *DAC* mediante:

$$DAC_{conv} = DAC_{res} * DAC_{val}$$

Nota

Para actualizar el valor de salida del *DAC* se utiliza la función hal_dac_update_value.

Velocidad de conversión

El periférico permite dos velocidades de conversión:

- Frecuencia de conversión de 1Mhz: Implica un mayor consumo de energía
- Frecuencia de conversión de 400KHz: Implica un menor consumo de energía

Nota

Existe una relación entre velocidad de conversión y consumo de energía del periférico. La velocidad de conversión del periférico, se configura en la función hal_dac_init.

Campos de aplicación típicos

- Generación de señales de naturaleza analógica
- Generación de señales auxiliares para calibración
- Servomecanismos controlados por señales analógicas

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)

  Inicialización del DAC.
- void hal_dac_update_value (hal_dac_en dac, uint16_t new_value)

  Actualización del valor actual del DAC.
- void hal_dac_config_ctrl (hal_dac_en dac, hal_dac_ctrl_config_t *config)

  Configuración del registro de control del DAC.

2.3.2. Documentación de las estructuras de datos

2.3.2.1. struct hal_dac_ctrl_config_t

Estructura de configuración del *DAC*

Campos de datos

uint8_t	count_enable: 1	Habilitación del contador (para modo DMA)
uint8_t	double_buffering: 1	Doble buffer (para modo DMA)
uint8_t	dma_enable: 1	Habilitacion de funcionamiento con el DMA
uint8_t	dma_request: 1	Pedido de DMA

2.3.3. Documentación de las enumeraciones

2.3.3.1. hal_dac_en

`enum hal_dac_en`

Enumeraciones de instancias disponibles de *DAC*

Valores de enumeraciones

HAL_DAC_0	Instancia 0
HAL_DAC_1	Instancia 1

2.3.3.2. hal_dac_settling_time_en

```
enum hal_dac_settling_time_en
```

Tiempos de establecimiento disponibles para el *DAC*

Valores de enumeraciones

HAL_DAC_SETTLING_TIME_1US_MAX	Tiempo de establecimiento de 1 microsegundo máximo
HAL_DAC_SETTLING_TIME_2_5US_MAX	Tiempo de establecimiento de 2.5 microsegundos máximo

2.3.4. Documentación de las funciones

2.3.4.1. hal_dac_init()

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

Inicialización del DAC.

Esta función inicializa el Control de Entrada/Salida (IOCON) de la forma necesaria para que el pin quede configurado correctamente.

Parámetros

in	*dac*	Cual de los dos DACs inicializar
in	*settling_time*	Velocidad de conversión del DAC
in	*initial_value*	Valor inicial del DAC

Ejemplos

Ejemplo_DAC.c.

2.3.4.2. hal_dac_update_value()

```
void hal_dac_update_value (
            hal_dac_en dac,
            uint16_t new_value )
```

Actualización del valor actual del DAC.

Parámetros

in	*dac*	En que DAC actualizar el valor
in	*new_value*	Nuevo valor a poner en el DAC

Precondición

Haber inicializado el periférico

Ejemplos

Ejemplo_DAC.c.

2.3.4.3. hal_dac_config_ctrl()

```
void hal_dac_config_ctrl (
            hal_dac_en dac,
            hal_dac_ctrl_config_t * config )
```

Configuración del registro de control del DAC.

Parámetros

in	*dac*	Que DAC configurar
in	*config*	Configuración deseada

Precondición

Haber inicializado el periférico

2.4. Entradas/Salidas de Propósito General (GPIO)

2.4.1. Descripción detallada

Introducción

El periférico *GPIO* es el encargado de controlar tanto las entradas como la salidas **digitales**. Esto implica que las salidas solamente podrán tomar valores *cero* o *uno* y que las entradas únicamente interpretarán valores *cero* o *uno*.

Cada uno de los pines del microcontrolador están descriptos mediante un número de *puerto* y un número de *pin*. En este caso se tienen 2 *puertos* (0 y 1) con 32 *pines* cada uno (0 a 31). Cabe destacar que si bien están definidas las condiciones para que se tengan un total de 64 pines de entrada/salida, la cantidad real disponible en el microcontrolador depende del encapsulado. En el caso del encapsulado trabajado en esta librería (por ahora), se tienen todos los pines del *puerto* 0, y los pines del 0 al 9 del *puerto* 1. Referirse a la sección Acerca del stick de desarrollo LPC845_BRK para más información.

Nota

Este periférico está intimamente relacionado con el periférico *IOCON*, particularmente para el manejo de ciertos aspectos de hardware de los pines. Para más información, referirse a la sección Control de Entrada/Salida (IOCON).

Funcionamiento de entradas

Cuando un pin en particular es configurado como *entrada*, se podrá leer en el mismo, el estado de una *señal digital externa*. Es importante que el usuario entienda que si la señal externa está en valores que correspondan a la *zona prohibida* de los niveles de señal digital, la lectura del valor será indefinida. En caso de necesitar leer valores *analógicos*, referirse al periférico Conversor analógico a digital (ADC) en esta misma documentación.

Nota

Las entradas son configuradas como tales mediante la función hal_gpio_set_dir y son consultadas/leídas mediante las funciones hal_gpio_read_pin y hal_gpio_read_port.

Funcionamiento de salidas

Cuando un pin en particular es configurado como *salida*, se podrán colocar en el mismo, *valores digitales*. Es importante que el usuario entienda que no hay forma de colocar un valor *analógico* en un pin mediante este periférico. Para dicha funcionalidad, referirse al periférico Conversor digital a analógico (DAC) en esta misma documentación.

Nota

Las salidas son configuradas como tales mediante la función hal_gpio_set_dir y son manipuladas individualmente mediante las funciones hal_gpio_set_pin, hal_gpio_clear_pin y hal_gpio_toggle_pin.

Enmascaramiento de entradas/salidas

El periférico *GPIO* dispone de la habilidad de *enmascarar* uno o más pines en particular. El hecho de *enmascarar* un pin implica que:

- Si el mismo estaba configurado como salida: Se inhabilita la posibilidad de cambiar el estado actual del pin.
- Si el mismo estaba configurado como entrada: Siempre se lee dicho pin como cero.

Nota

Las máscaras son configuradas mediante las funciones hal_gpio_set_mask_bits, hal_gpio_lear_mask_bits y hal_gpio_toggle_mask_bits.

Lectura/Escritura de Puerto/Pin

Al usuario puede serle de utilidad la posibilidad de leer/escribir el estado de un pin en particular, así como también la posibilidad de leer/escribir varios pines a la vez. Si se encuentra en el segundo caso, el periférico *GPIO* dispone de la habilidad de hacerlo, siempre y cuando los múltiples pines se encuentren **en el mismo puerto**.

Es importante también destacar, que las lecturas/escrituras respetarán las máscaras explicadas anteriormente, por lo cual es recomendable configurar dicha funcionalidad a la hora de leer/escribir múltiples pines, y asimismo es recomendable "liberar" los enmascaramientos una vez trabajados los pines necesarios.

Nota

Las funciones que involucran máscaras son hal_gpio_masked_set_port, hal_gpio_masked_clear_port, hal_gpio_masked_toggle_port y hal_gpio_masked_read_port.

Campos de aplicación típicos

- Lectura de estado de variables externas digitales (switch, sensor digital)
- Escritura para accionar variables externas digitales (LED, motores)

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_PORTPIN_0_27,
  HAL_GPIO_PORTPIN_0_28,
  HAL_GPIO_PORTPIN_0_29,
  HAL_GPIO_PORTPIN_0_30,
  HAL_GPIO_PORTPIN_0_31,
  HAL_GPIO_PORTPIN_1_0,
  HAL_GPIO_PORTPIN_1_1,
  HAL_GPIO_PORTPIN_1_2,
  HAL_GPIO_PORTPIN_1_3,
  HAL_GPIO_PORTPIN_1_4,
  HAL_GPIO_PORTPIN_1_5,
  HAL_GPIO_PORTPIN_1_6,
  HAL_GPIO_PORTPIN_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 dirección de una GPIO.
- void hal_gpio_set_pin (hal_gpio_portpin_en portpin)

  Fijar estado de una GPIO (sin importar máscara)
- void hal_gpio_set_port (hal_gpio_port_en port, uint32_t bits_to_set)

  Fijar estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_set_port (hal_gpio_port_en port, uint32_t bits_to_set)

  Fijar estado de pines de un puerto (teniendo en cuenta máscara)
- void hal_gpio_clear_pin (hal_gpio_portpin_en portpin)

  Limpiar estado de una GPIO (sin importar máscara)
- void hal_gpio_clear_port (hal_gpio_port_en port, uint32_t bits_to_clear)

  Limpiar estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_clear_port (hal_gpio_port_en port, uint32_t bits_to_clear)

  Limpiar estado de pines de un puerto (teniendo en cuenta máscara)
- void hal_gpio_toggle_pin (hal_gpio_portpin_en portpin)

  Invertir estado de una GPIO (sin importar máscara)
- void hal_gpio_toggle_port (hal_gpio_port_en port, uint32_t bits_to_toggle)

  Invertir estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_toggle_port (hal_gpio_port_en port, uint32_t bits_to_toggle)

  Invertir estado de pines de un puerto (teniendo en cuenta máscara)
- uint8_t hal_gpio_read_pin (hal_gpio_portpin_en portpin)

  Leer el estado de una GPIO (sin importar máscara)
- uint32_t hal_gpio_read_port (hal_gpio_port_en port)

  Leer estado de un puerto (sin importar máscara)
- uint32_t hal_gpio_masked_read_port (hal_gpio_port_en port)

  Leer estado de un puerto (teniendo en cuenta máscara)
- void hal_gpio_set_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Fijar enmascaramiento de pines en un puerto.
- void hal_gpio_clear_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Limpiar enmascaramiento de pines en un puerto.
- void hal_gpio_toggle_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Invertir enmascaramiento de pines en un puerto.

2.4.2. Documentación de los 'defines'

2.4.2.1. HAL_GPIO_PORTPIN_TO_PORT

```
#define HAL_GPIO_PORTPIN_TO_PORT(
              x ) (x / 32)
```

Macro para obtener número de puerto a partir de un puerto/pin

Ejemplos

Ejemplo_GPIO.c.

2.4.2.2. HAL_GPIO_PORTPIN_TO_PIN

```
#define HAL_GPIO_PORTPIN_TO_PIN(
            x ) (x % 32)
```

Macro para obtener número de pin a partir de un puerto/pin

Ejemplos

Ejemplo_GPIO.c.

2.4.3. Documentación de las enumeraciones

2.4.3.1. hal_gpio_port_en

```
enum hal_gpio_port_en
```

Enumeraciones de puertos disponibles

Valores de enumeraciones

HAL_GPIO_PORT↩ _0	Puerto 0
HAL_GPIO_PORT↩ _1	Puerto 1

2.4.3.2. hal_gpio_portpin_en

```
enum hal_gpio_portpin_en
```

Enumeraciones de puerto/pin disponibles

Valores de enumeraciones

HAL_GPIO_PORTPIN_0_0	Puerto 0, pin 0
HAL_GPIO_PORTPIN_0_1	Puerto 0, pin 1
HAL_GPIO_PORTPIN_0_2	Puerto 0, pin 2
HAL_GPIO_PORTPIN_0_3	Puerto 0, pin 3
HAL_GPIO_PORTPIN_0_4	Puerto 0, pin 4
HAL_GPIO_PORTPIN_0_5	Puerto 0, pin 5
HAL_GPIO_PORTPIN_0_6	Puerto 0, pin 6
HAL_GPIO_PORTPIN_0_7	Puerto 0, pin 7
HAL_GPIO_PORTPIN_0_8	Puerto 0, pin 8
HAL_GPIO_PORTPIN_0_9	Puerto 0, pin 9

Valores de enumeraciones

HAL_GPIO_PORTPIN_0_10	Puerto 0, pin 10
HAL_GPIO_PORTPIN_0_11	Puerto 0, pin 11
HAL_GPIO_PORTPIN_0_12	Puerto 0, pin 12
HAL_GPIO_PORTPIN_0_13	Puerto 0, pin 13
HAL_GPIO_PORTPIN_0_14	Puerto 0, pin 14
HAL_GPIO_PORTPIN_0_15	Puerto 0, pin 15
HAL_GPIO_PORTPIN_0_16	Puerto 0, pin 16
HAL_GPIO_PORTPIN_0_17	Puerto 0, pin 17
HAL_GPIO_PORTPIN_0_18	Puerto 0, pin 18
HAL_GPIO_PORTPIN_0_19	Puerto 0, pin 19
HAL_GPIO_PORTPIN_0_20	Puerto 0, pin 20
HAL_GPIO_PORTPIN_0_21	Puerto 0, pin 21
HAL_GPIO_PORTPIN_0_22	Puerto 0, pin 22
HAL_GPIO_PORTPIN_0_23	Puerto 0, pin 23
HAL_GPIO_PORTPIN_0_24	Puerto 0, pin 24
HAL_GPIO_PORTPIN_0_25	Puerto 0, pin 25
HAL_GPIO_PORTPIN_0_26	Puerto 0, pin 26
HAL_GPIO_PORTPIN_0_27	Puerto 0, pin 27
HAL_GPIO_PORTPIN_0_28	Puerto 0, pin 28
HAL_GPIO_PORTPIN_0_29	Puerto 0, pin 29
HAL_GPIO_PORTPIN_0_30	Puerto 0, pin 30
HAL_GPIO_PORTPIN_0_31	Puerto 0, pin 31
HAL_GPIO_PORTPIN_1_0	Puerto 1, pin 0
HAL_GPIO_PORTPIN_1_1	Puerto 1, pin 1
HAL_GPIO_PORTPIN_1_2	Puerto 1, pin 2
HAL_GPIO_PORTPIN_1_3	Puerto 1, pin 3
HAL_GPIO_PORTPIN_1_4	Puerto 1, pin 4
HAL_GPIO_PORTPIN_1_5	Puerto 1, pin 5
HAL_GPIO_PORTPIN_1_6	Puerto 1, pin 6
HAL_GPIO_PORTPIN_1_7	Puerto 1, pin 7
HAL_GPIO_PORTPIN_1_8	Puerto 1, pin 8
HAL_GPIO_PORTPIN_1_9	Puerto 1, pin 9
HAL_GPIO_PORTPIN_NOT_USED	Puerto/pin no utilizado

2.4.3.3. hal_gpio_dir_en

`enum hal_gpio_dir_en`

Enumeraciones de direcciones posibles para un puerto/pin

Valores de enumeraciones

HAL_GPIO_DIR_INPUT	Puerto/pin como entrada
HAL_GPIO_DIR_OUTPUT	Puerto/pin como salida

2.4.4. Documentación de las funciones

2.4.4.1. hal_gpio_init()

```
void hal_gpio_init (
            hal_gpio_port_en port )
```

Inicializar un puerto.

Parámetros

in	*port*	Puerto a inicializar

Ejemplos

Ejemplo_ADC.c, Ejemplo_GPIO.c, Ejemplo_PININT.c y Ejemplo_WKT.c.

2.4.4.2. hal_gpio_set_dir()

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

Fijar dirección de una GPIO.

Parámetros

in	*portpin*	Número de puerto/pin a configurar
in	*dir*	Dirección deseada
in	*initial_state*	Estado inicial (aplica únicamente para salidas)

Nota

Es importante recordar que el Control de Entrada/Salida (IOCON) controla aspectos de hardware del puerto/pin, por ejemplo la inversión en la lógica del mismo, y que esta función no configura ninguno de esos aspectos. En caso de ser necesario configurar dichas características, ver hal_iocon_config_io.

Ver también

hal_gpio_portpin_en

hal_gpio_dir_en

Precondición

Haber inicializado el puerto correspondiente

Parámetros

in	*portpin*	Número de puerto/pin a configurar
in	*dir*	Dirección deseada
in	*initial_state*	Estado inicial (aplica únicamente para salidas)

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_ADC.c, Ejemplo_GPIO.c, Ejemplo_PININT.c y Ejemplo_WKT.c.

2.4.4.3. hal_gpio_set_pin()

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

Fijar estado de una GPIO (sin importar máscara)

Parámetros

in	*portpin*	Número de puerto/pin a accionar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_ADC.c.

2.4.4.4. hal_gpio_set_port()

```
void hal_gpio_set_port (
            hal_gpio_port_en port,
            uint32_t bits_to_set )
```

Fijar estado de pines de un puerto (sin importar máscara)

Parámetros

in	*port*	Número de puerto a accionar
in	*bits_to_set*	Máscara de bits a fijar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.5. hal_gpio_masked_set_port()

```
void hal_gpio_masked_set_port (
            hal_gpio_port_en port,
            uint32_t bits_to_set )
```

Fijar estado de pines de un puerto (teniendo en cuenta máscara)

Parámetros

in	*port*	Número de puerto a accionar
in	*bits_to_set*	Máscara de bits a fijar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.6. hal_gpio_clear_pin()

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

Limpiar estado de una GPIO (sin importar máscara)

Parámetros

in	*portpin*	Número de puerto/pin a accionar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_ADC.c.

2.4.4.7. hal_gpio_clear_port()

```
void hal_gpio_clear_port (
            hal_gpio_port_en port,
            uint32_t bits_to_clear )
```

Limpiar estado de pines de un puerto (sin importar máscara)

Parámetros

in	*port*	Número de puerto a accionar
in	*bits_to_clear*	Máscara de bits a limpiar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.8. hal_gpio_masked_clear_port()

```
void hal_gpio_masked_clear_port (
            hal_gpio_port_en port,
            uint32_t bits_to_clear )
```

Limpiar estado de pines de un puerto (teniendo en cuenta máscara)

Parámetros

in	*port*	Número de puerto a accionar
in	*bits_to_clear*	Máscara de bits a limpiar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.9. hal_gpio_toggle_pin()

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

Invertir estado de una GPIO (sin importar máscara)

Parámetros

in	*portpin*	Número de puerto/pin a accionar

Ver también

hal_gpio_portpin_en

Parámetros

in	*portpin*	Número de puerto/pin a accionar

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_WKT.c.

2.4.4.10. hal_gpio_toggle_port()

```
void hal_gpio_toggle_port (
            hal_gpio_port_en port,
            uint32_t bits_to_toggle )
```

Invertir estado de pines de un puerto (sin importar máscara)

Parámetros

in	*port*	Número de puerto a accionar
in	*bits_to_toggle*	Máscara de bits a invertir

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.11. hal_gpio_masked_toggle_port()

```
void hal_gpio_masked_toggle_port (
            hal_gpio_port_en port,
            uint32_t bits_to_toggle )
```

Invertir estado de pines de un puerto (teniendo en cuenta máscara)

Parámetros

in	*port*	Número de puerto a accionar
in	*bits_to_toggle*	Máscara de bits a invertir

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_GPIO.c.

2.4.4.12. hal_gpio_read_pin()

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

Leer el estado de una GPIO (sin importar máscara)

Parámetros

in	*portpin*	Número de puerto/pin a accionar

Devuelve

Estado actual del puerto/pin omitiendo máscara

Ver también

hal_gpio_portpin_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_GPIO.c.

2.4.4.13. hal_gpio_read_port()

```
uint32_t hal_gpio_read_port (
            hal_gpio_port_en port )
```

Leer estado de un puerto (sin importar máscara)

Parámetros

in	*port*	Puerto a consultar

Devuelve

Estado del puerto sin enmascarar

Ver también

hal_gpio_port_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.14. hal_gpio_masked_read_port()

```
uint32_t hal_gpio_masked_read_port (
            hal_gpio_port_en port )
```

Leer estado de un puerto (teniendo en cuenta máscara)

Parámetros

in	*port*	Puerto a consultar

Devuelve

Estado del puerto contemplando la máscara asociada

Ver también

hal_gpio_port_en

Precondición

Haber inicializado el puerto correspondiente

2.4.4.15. hal_gpio_set_mask_bits()

```
void hal_gpio_set_mask_bits (
            hal_gpio_port_en port,
            uint32_t mask )
```

Fijar enmascaramiento de pines en un puerto.

Parámetros

in	*port*	Puerto a fijar enmascaramiento
in	*mask*	Máscara de bits a fijar

Ver también

hal_gpio_port_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_GPIO.c.

2.4.4.16. hal_gpio_clear_mask_bits()

```
void hal_gpio_clear_mask_bits (
            hal_gpio_port_en port,
            uint32_t mask )
```

Limpiar enmascaramiento de pines en un puerto.

Parámetros

in	*port*	Puerto a limpiar enmascaramiento
in	*mask*	Máscara de bits a limpiar

Ver también

hal_gpio_port_en

Precondición

Haber inicializado el puerto correspondiente

Ejemplos

Ejemplo_GPIO.c.

2.4.4.17. hal_gpio_toggle_mask_bits()

```
void hal_gpio_toggle_mask_bits (
            hal_gpio_port_en port,
            uint32_t mask )
```

Invertir enmascaramiento de pines en un puerto.

Parámetros

in	*port*	Puerto a invertir enmascaramiento
in	*mask*	Máscara de bits a invertir

Ver también

hal_gpio_port_en

Precondición

Haber inicializado el puerto correspondiente

2.5. Control de Entrada/Salida (IOCON)

2.5.1. Descripción detallada

Introducción

El periférico *IOCON* es el responsable de configurar las características eléctricas de cada pin del microcontrolador. Dichas características son:

- Resistor de Pull-up/Pull-Down
- Modo *open rain*
- Histéresis
- Filtro de *glitches* configurable
- Modo analógico
- Modo IIC *Fast Mode - Plus* en una de las interfaces

Inversión de lógica del pin

Este periférico tiene la posibilidad de invertir la lógica del pin, cuando el mismo es una entrada, mediante hardware. Asumiendo que no está activada la inversión del pin y que está configurado como entrada, si se toma una lectura del mismo, se leerá como **cero** cuando externamente esté a una tensión de aproximadamente VSS y se leerá como **uno** cuando externamente esté a una tensión de aproximadamente VDD. Si se activa la inversión del pin sucederá lo inverso, es decir, si está configurado como entrada y externamente hay una tensión de aproximadamente VSS la lectura se tomará como **uno** y si hay una tensión de aproximadamente VDD la lectura se tomará como **cero**.

Resistencias para fijar un nivel

Cada uno de los pines del microcontrolador puede ser configurado para tener una resistencia de *pull-up* o de *pull-down*. Esto implica que aunque la entrada no tenga ningún valor fuerte externamente, estas resistencias se encargarán de fijar un valor o bien naturalmente alto (*pull-up*) o bajo (*pull-down*). Existe un tercer tipo llamado *repetidor* el cual configura un *pull-up* o un *pull-down* dependiendo del último valor que hubo en la entrada, esto quiere decir que si el pin tiene un valor externamente fijado en estado alto, y queda sin un valor fijo, el microcontrolador automáticamente dejará configurado un *pull-up*, de esta forma reteniendo el último valor fijo, y viceversa cuando el pin es fijado externamente en estado bajo.

Salidas de tipo *Open Drain*

Una salida tipo *open drain* en comparación con una salida tradicional, permite colocar un nivel de tensión distinto al de la alimentación del microcontrolador cuando se acciona el pin.

NOTA: La hoja de datos aclara que la tensión a la cual se puede manejar el pin mediante un resistor de *pull-up* externo, no puede superar la tensión VDD.

Histéresis

Cada pin puede configurar una *histéresis* cuando se comporta como entrada. Que el pin tenga una *histéresis* implica que el valor cambiará su lectura una vez superado un cierto umbral Vhys. Esto evita lecturas erróneas cuando la entrada tiene una variación a un ritmo lenta que no se puede evitar.

Filtro de *Glitches*

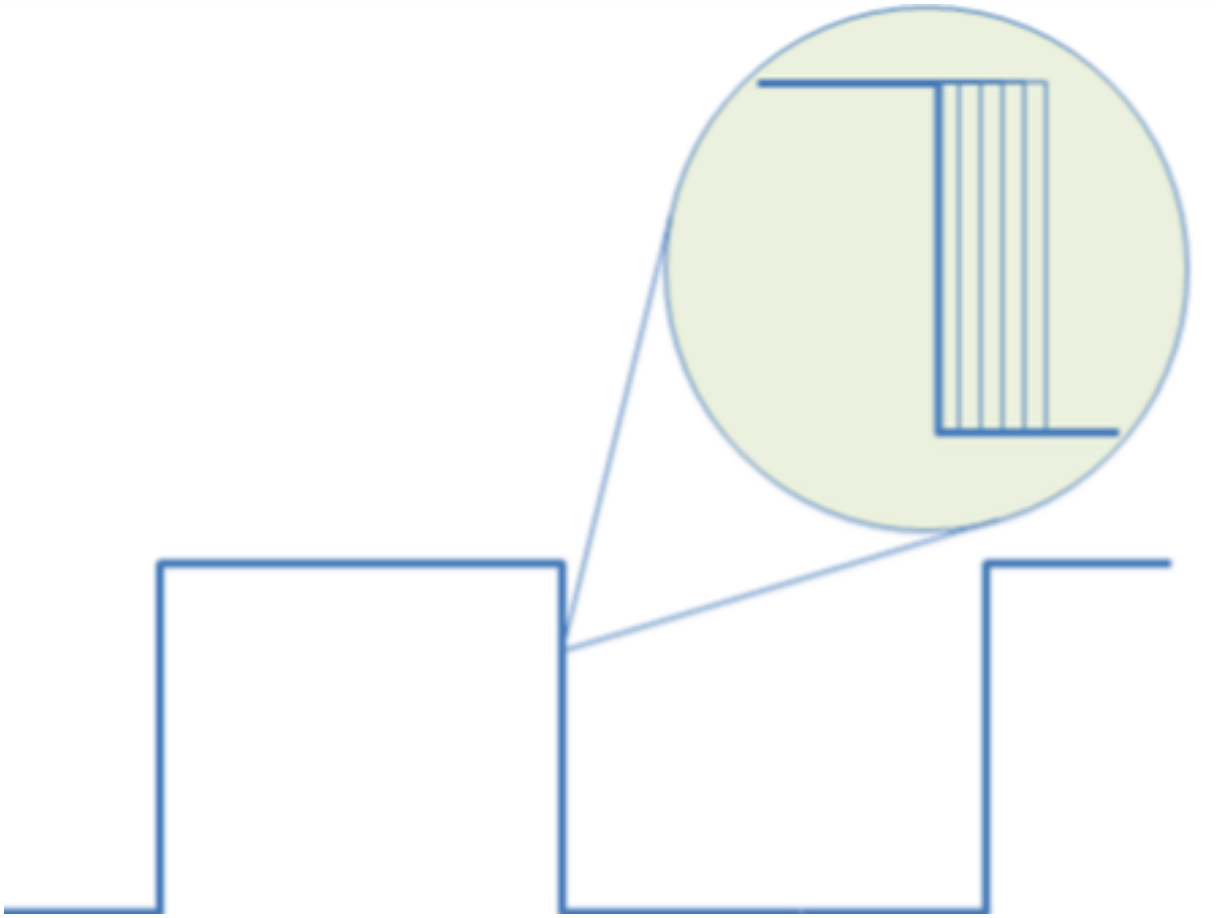


Figura 2.1 Entrada con *Glitches* width

Si una entrada tiene \emph{glitches}, como se muestra en la imágen, se pueden tomar lecturas erróneas a causa de la falta de estabilidad de la señal que excita a la entrada. Una forma de evitar este problema es configurando correctamente el clock asociado al periférico *IOCON* mediante el Configuración del Sistema (SYSCON), y el filtro de *glitches* asociado al pin, el cual tomará una lectura como válida luego de una cantidad configurable de ciclos del clock mencionado.

Modo analógico

Para utilizar un pin como salida analógica, como se describe en el periférico Conversor digital a analógico (DAC), se debe configurar dicho pin como salida analógica también mediante el periférico *IOCON*. Además se debe inhabilitar cualquier tipo de resistor interno cuando se configura el pin como salida analógica.

Modo IIC

Cuando se utiliza el periférico IIC, en particular la instancia 0, se debe configurar mediante el *IOCON* dependiendo de la velocidad con la que se va a utilizar.

Implementación en la librería

Es importante destacar que cada periférico inicializará el *IOCON* en el punto que lo necesite, y luego lo desactivará para conservar energía. La única excepción es el periférico Entradas/Salidas de Propósito General (GPIO) el cual no genera ninguna inicialización respecto del *IOCON*, por lo que el usuario deberá configurar (en caso de ser necesario) cada uno de los pines como se desee.

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.

2.5.2. Documentación de las estructuras de datos

2.5.2.1. struct hal_iocon_config_t

Estructura de configuración de un pin mediante el *IOCON*

Ejemplos

Ejemplo_GPIO.c.

Campos de datos

hal_iocon_pull_mode_en	pull_mode	Resistor interno
uint8_t	hysteresis	Histéresis. Cualquier valor distinto de cero la activa
uint8_t	invert_input	Inversión de lógica de lectura. Cualquier valor distinto de cero la activa
uint8_t	open_drain	Modo *Open Drain*. Cualquier valor distinto de cero lo activa
hal_iocon_sample_mode_en	sample_mode	Cantidad de muestras del filtro de glitches
hal_iocon_clk_sel_en	clk_sel	Selección de clock para el filtro de glitches
hal_iocon_iic_mode_en	iic_mode	Selección de modo IIC

2.5.3. Documentación de las enumeraciones

2.5.3.1. hal_iocon_pull_mode_en

`enum hal_iocon_pull_mode_en`

Selección de resistor interno en el pin

Valores de enumeraciones

HAL_IOCON_PULL_NONE	Ningun resistor interno
HAL_IOCON_PULL_DOWN	Resistor interno conectado a Vss
HAL_IOCON_PULL_UP	Resistor interno conectado a Vdd
HAL_IOCON_PULL_REPEATER	Modo repetidor

2.5.3.2. hal_iocon_sample_mode_en

`enum hal_iocon_sample_mode_en`

Modo de muestreo del filtro de glitches

Valores de enumeraciones

HAL_IOCON_SAMPLE_MODE_BYPASS	Sin filtro de glitches
HAL_IOCON_SAMPLE_MODE_1_CLOCK	Por lo menos un ciclo de clock para filtrar el glitch
HAL_IOCON_SAMPLE_MODE_2_CLOCK	Por lo menos dos ciclos de clock para filtrar el glitch
HAL_IOCON_SAMPLE_MODE_3_CLOCK	Por lo menos tres ciclos de clock para filtrar el glitch

2.5.3.3. hal_iocon_clk_sel_en

```
enum hal_iocon_clk_sel_en
```

Selección de banco de división de clock para el filtro de glitches

Valores de enumeraciones

HAL_IOCON_CLK_DIV↩ _0	Banco 0
HAL_IOCON_CLK_DIV↩ _1	Banco 1
HAL_IOCON_CLK_DIV↩ _2	Banco 2
HAL_IOCON_CLK_DIV↩ _3	Banco 3
HAL_IOCON_CLK_DIV↩ _4	Banco 4
HAL_IOCON_CLK_DIV↩ _5	Banco 5
HAL_IOCON_CLK_DIV↩ _6	Banco 6

2.5.3.4. hal_iocon_iic_mode_en

```
enum hal_iocon_iic_mode_en
```

Modo de funcionamiento IIC

Valores de enumeraciones

HAL_IOCON_IIC_MODE_STANDARD	Modo IIC standard
HAL_IOCON_IIC_MODE_GPIO	Modo GPIO
HAL_IOCON_IIC_MODE_FAST_MODE	Modo IIC Fast-Mode

2.5.4. Documentación de las funciones

2.5.4.1. hal_iocon_config_io()

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

Configuracion de un pin.

Nótese que las configuraciones de modo analógico no están en la estructura de configuración hal_iocon_config_t. Esto es debido a que dichas configuraciones se realizarán en el periférico correspondiente, sea el Conversor analógico a digital (ADC) o Conversor digital a analógico (DAC).

Parámetros

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

Precondición

Configuración de divisores de clock de bancos de filtro de glitches en caso de ser necesario.

Parámetros

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

Ejemplos

Ejemplo_GPIO.c.

2.6. Interrupciones de pin / Motor de patrones (PININT)

2.6.1. Descripción detallada

Introducción

El periférico *PININT* puede funcionar tanto como *Interrupciones de pin* como *Motor de detección de patrones*.

Funcionamiento como *Interrupciones de pin*

El periférico dispone de 8 canales configurables para detección de distintos eventos externos al microcontrolador. Cada uno de estos canales es configurable para detectar cambios de nivel o de flanco en el pin asociado, pudiendo así también detectar flancos ascendentes/descendentes como nivel alto/bajo. Cada canal tiene su propia configuración independiente de los demás.

Funcionamiento como *Motor de detección de patrones*

Atención

Falta desarrollar!

Campos de aplicación típicos

- Detección de eventos externos que necesiten prioridad
- Utilización como interfaz para disparar otros periféricos, como el Conversor analógico a digital (ADC), mediante señales externas
- Detección de patrones en múltiples pines externos

typedefs

- typedef void(∗ hal_pinint_callback_t) (void)
  Tipo de dato para callback de PININT.

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_edge_detections_en {
  HAL_PININT_EDGE_DETECTIONS_NONE = 0,
  HAL_PININT_EDGE_DETECTIONS_RISING,
  HAL_PININT_EDGE_DETECTIONS_FALLING,
  HAL_PININT_EDGE_DETECTIONS_BOTH }
- enum hal_pinint_level_detections_en {
  HAL_PININT_LEVEL_DETECTIONS_NONE = 0,
  HAL_PININT_LEVEL_DETECTIONS_HIGH,
  HAL_PININT_LEVEL_DETECTIONS_LOW }

Funciones

- void hal_pinint_init (void)

  Inicialización del periférico.
- void hal_pinint_deinit (void)

  De-Inicialización del periférico.
- void hal_pinint_channel_config (hal_pinint_channel_en channel, hal_gpio_portpin_en portpin, hal_pinint_callback_t callback)

  Configuración de canal de PININT.
- void hal_pinint_edge_detections_config (hal_pinint_channel_en channel, hal_pinint_edge_detections_en edge)

  Configurar detecciones por flanco.
- void hal_pinint_level_detections_config (hal_pinint_channel_en channel, hal_pinint_level_detections_en level)

  Configurar detecciones por nivel.

2.6.2. Documentación de los 'typedefs'

2.6.2.1. hal_pinint_callback_t

```
typedef void(* hal_pinint_callback_t) (void)
```

Tipo de dato para callback de *PININT*.

Nota

Cabe recordar que estos callbacks se ejecutan bajo el contexto de una interrupción, por lo que el mismo deberá tener todas las consideraciones necesarias

2.6.3. Documentación de las enumeraciones

2.6.3.1. hal_pinint_channel_en

```
enum hal_pinint_channel_en
```

Seleccion de canal de *PININT*

Valores de enumeraciones

HAL_PININT_CHANNEL↩_0	Canal 0
HAL_PININT_CHANNEL↩_1	Canal 1
HAL_PININT_CHANNEL↩_2	Canal 2

Valores de enumeraciones

HAL_PININT_CHANNEL↩_3	Canal 3
HAL_PININT_CHANNEL↩_4	Canal 4
HAL_PININT_CHANNEL↩_5	Canal 5
HAL_PININT_CHANNEL↩_6	Canal 6
HAL_PININT_CHANNEL↩_7	Canal 7

2.6.3.2. hal_pinint_edge_detections_en

`enum hal_pinint_edge_detections_en`

Selección de flancos a detectar

Valores de enumeraciones

HAL_PININT_EDGE_DETECTIONS_NONE	Inhabiltar detecciones por flanco
HAL_PININT_EDGE_DETECTIONS_RISING	Detección de flancos ascendentes
HAL_PININT_EDGE_DETECTIONS_FALLING	Detección de flancos descendentes
HAL_PININT_EDGE_DETECTIONS_BOTH	Detección de ambos flancos

2.6.3.3. hal_pinint_level_detections_en

`enum hal_pinint_level_detections_en`

Selección de polaridad para detecciones por nivel

Valores de enumeraciones

HAL_PININT_LEVEL_DETECTIONS_NONE	Inhabilitar detecciones por nivel
HAL_PININT_LEVEL_DETECTIONS_HIGH	Polaridad positiva
HAL_PININT_LEVEL_DETECTIONS_LOW	Polaridad negativa

2.6.4. Documentación de las funciones

2.6.4.1. hal_pinint_init()

```
void hal_pinint_init (
            void  )
```

Inicialización del periférico.

Ejemplos

Ejemplo_PININT.c.

2.6.4.2. hal_pinint_deinit()

```
void hal_pinint_deinit (
            void  )
```

De-Inicialización del periférico.

2.6.4.3. hal_pinint_channel_config()

```
void hal_pinint_channel_config (
            hal_pinint_channel_en channel,
            hal_gpio_portpin_en portpin,
            hal_pinint_callback_t callback )
```

Configuración de canal de *PININT*.

Nota

Esta función no configura el modo de detección. Ver: hal_pinint_edge_detections_config y hal_pinint_level_detections_config

Parámetros

in	*channel*	Canal a configurar
in	*portpin*	Puerto/pin en donde configurar el canal
in	*callback*	Callback a ejecutar en detección

Precondición

Haber inicializado el periférico

Ejemplos

Ejemplo_PININT.c.

2.6.4.4. hal_pinint_edge_detections_config()

```
void hal_pinint_edge_detections_config (
            hal_pinint_channel_en channel,
            hal_pinint_edge_detections_en edge )
```

Configurar detecciones por flanco.

Parámetros

in	*channel*	Canal a configurar
in	*edge*	Flancos a detectar

Precondición

Haber inicializado el periférico

2.6.4.5. hal_pinint_level_detections_config()

```
void hal_pinint_level_detections_config (
            hal_pinint_channel_en channel,
            hal_pinint_level_detections_en level )
```

Configurar detecciones por nivel.

Parámetros

in	*channel*	Canal a configurar
in	*level*	Nivel a detectar

Precondición

Haber inicializado el periférico

Ejemplos

Ejemplo_PININT.c.

2.7. Configuración del Sistema (SYSCON)

2.7.1. Descripción detallada

Introducción

Este periférico es el encargado de manejar múltiples características del sistema del microcontrolador. Entre ellas se encuentran:

- Control de clocks
- Control de reset/clocks de periféricos
- Selección de pines para interrupciones externas (Ver Interrupciones de pin / Motor de patrones (PININT))
- Configuración de modos de bajo consumo del microcontrolador
- Configuración de wake-up para salir de los modos de bajo consumo del microcontrolador
- Configuración del Brown-Out Detector
- Configuración del Micro Trace Buffer
- Control de interrupción de latencias
- Control de Non Maskable Interrupt
- Valor de calibración del Tick del Sistema (SYSTICK)

En la capa de abstracción de aplicación (HAL) se implementan principalmente las funciones relacionadas con los tres primeros ítems hasta el momento.

Control de clocks

Fuentes de clock

Las distintas fuentes de clock que se explican a continuación, son seleccionables mediante el *SYSCON*. Algunos necesitan una referencia para funcionar, mientras que otros funcionan sin necesidad de ninguna.

Free Running Oscillator (FRO) Este oscilador es con el que comienza el microcontrolador por defecto luego de un reset. La frecuencia del mismo se puede configurar (*no implementado todavia*) pero por default comienza en 24MHz con un divisor /2, resultando en una frecuencia efectiva de 12MHz.

Nota

Las funciones relacionadas con el *FRO* son hal_syscon_fro_clock_config y hal_syscon_fro_clock_get

Phase Locked Loop (PLL) Este oscilador toma una frecuencia de entrada y genera una conversión para obtener una frecuencia efectiva mayor a la de entrada. La frecuencia de entrada mínima del mismo es de 10MHz.

Nota

Las funciones relacionadas con el *PLL* son hal_syscon_pll_clock_config y hal_syscon_pll_clock_get.

Main/System clock El clock principal o de sistema (estos nombres se usan indistintamente) genera la frecuencia base de la cual se derivan la mayoría de los periféricos. El mismo puede ser tomado de la señal de salida del *PLL* o de la señal previa al *PLL*. Este clock es el que provee la frecuencia del núcleo del microcontrolador.

Nota

La selección de la fuente de clock principal se realiza mediante la función hal_syscon_system_clock_set_source mientras que la obtención de la frecuencia actual del clock principal configurada se obtiene con la función hal_syscon_system_clock_get.

Clock externo El clock externo puede ser de utilidad cuando se tiene una referencia de frecuencia de muy buena estabilidad externa al microcontrolador. Casos típicos son *cristales*, u osciladores de alta presición y bajo drift. Si se utiliza un *cristal externo* se utilizarán los pines P0_8 y P0_9 como fuentes de entrada para el circuito oscilador interno, el cual se encargará de generar la frecuencia de clock correspondiente, mientras que si se utiliza un oscilador externo, se utilizará únicamente el pin P0_1.

Nota

Para configurar el clock externo se utilizan las funciones hal_syscon_external_crystal_config y hal_syscon_external_clock_config.

Generadores fraccionales de clock El microcontrolador dispone de dos *Generadores fraccionales de clock*. Los mismos son de gran utilidad cuando se necesita tener presición en la frecuencia de algún periférico y la frecuencia del clock principal con los divisores del periférico no nos alcanzas para lograr dicha presición. La ventaja de estos generadores es que toman una frecuencia de referencia, y generan una división fraccional del mismo. El divisor de estos generadores varía entre 1 y 2 con valores decimales, pudiendo así lograr frecuencias que los demás divisores no pueden, dada su naturaleza de división entera.

Periféricos que pueden ser excitados mediante estos generadores:

- UART
- SPI
- IIC

Nota

La función para configurar los generadores fraccionales de clock es hal_syscon_frg_config.

Oscilador del *Watchdog* El periférico WATCHDOG tiene su propia fuente de clock. Este oscilador es de ultra bajo consumo, pero su presición es de más/menos 40. El oscilador puede tener como base una variedad de valores y también tiene su propio divisor, logrando rangos entre 9.3KHz y 2.3MHz. Se parte de una frecuencia base seleccionable, y luego se aplica un divisor al mismo para obtener la frecuencia final de oscilación.

Nota

Las características de este oscilador se controlan mediante la función hal_syscon_watchdog_oscillator_config

Divisores de clock

Ciertos periféricos permiten dividir la frecuencia de clock principal o de otra fuente, para así tener un menor consumo de energía, o para cumplir con especificaciones de frecuencia que requiera el periférico particularmente.

Divisor del clock principal La frecuencia del clock principal puede ser dividida por cualquier número entero entre 1 y 255. Para situaciones donde la velocidad de procesamiento no sea una necesidad primordial, se puede reducir la misma mediante este divisor. Como este divisor afecta al clock principal, y la mayoría de los periféricos corren con un clock asociado al mismo, se verá reducido el consumo notablemente, a expensas de reducir la velocidad de procesamiento.

Nota

La función para el control de este divisor es hal_syscon_system_clock_set_divider.

Divisor del clock del ADC El clock de la lógica del periférico Conversor analógico a digital (ADC) es alimentado por el clock principal luego de pasar por este divisor. El divisor puede ser configurado en cualquier valor entero entre 0 y 255. Si se coloca en 0, el clock del *ADC* será anulado. Nótese que la configuración necesaria del divisor es realizada en las funciones de inicialización del *ADC*.

Nota

El periférico Conversor analógico a digital (ADC) se ocupará de configurar su divisor en caso de ser necesario. No se proveen funciones en este módulo para la configuración del mismo.

Divisor del clock del SCT Al igual que con el *ADC*, el clock de la lógica del periférico SCT es alimentado por el clock principal luego de pasar por este divisor. El divisor puede ser configurado en cualquier valor entero entre 0 y 255. Si se coloca en 0, el clock del *SCT* será anulado.

Nota

Este periférico, así como las funciones para configurar su divisor en el *SYSCON*, todavía no está implementado en la librería.

Divisor de la salida CLKOUT En caso de ser necesario, el periférico *SYSCON* puede ser configurado para general una salida en uno de los pines del microcontrolador que esté relacionada a algunas fuentes de clock. Antes de salir por el pin, la señal pasa por un divisor, el cual puede ser configurado en cualquier valor entero entre 0 y 255. Si el mismo es configurado en 0, la salida es anulada. Fuentes de clock disponibles para la salida CLKOUT:

- FRO
- Clock principal
- PLL
- Clock externo (Cristal/entrada externa)
- Oscilador del watchdog

Nota

La función para el manejo de la salida *CLKOUT* es hal_syscon_clkout_config.

Divisores para el filtro de Glitches del IOCON El periférico Control de Entrada/Salida (IOCON) tiene la posibilidad de ser configurado para eliminar glitches en entradas mediante un filtrado por hardware. Dicho filtro de glitches puede ser configurado para tomar su señal de excitación de uno de los siete divisores que tiene el *SYSCON* reservados para este fin. El valor de estos divisores puede ser cualquier valor entero entre 0 y 255. Si se configura como 0, el divisor no generará señal de excitación, anulando así la funcionalidad.

Nota

La función para el manejo de los divisores de filtros de glitches es hal_syscon_iocon_glitch_divider_set.

Enumeraciones

- enum hal_syscon_system_clock_sel_en {
  HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO = 0,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_EXT,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_WATCHDOG,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO_DIV,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_PLL }
- 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_watchdog_clkana_sel_en {
  HAL_SYSCON_WATCHDOG_CLKANA_0KHZ = 0,
  HAL_SYSCON_WATCHDOG_CLKANA_600KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_1050KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_1400KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_1750KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_2100KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_2400KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_3000KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_3250KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_3500KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_3750KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_4000KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_4200KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_4400KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_4600KHZ }
- 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_system_clock_get (void)
  Obtener la frecuencia actual del main clock.
- void hal_syscon_system_clock_set_source (hal_syscon_system_clock_sel_en clock_source)
  Configuración de fuente de clock para el clock principal.
- void hal_syscon_system_clock_set_divider (uint8_t div)
  Fijar el divisor del clock principal.
- uint32_t hal_syscon_fro_clock_get (void)
  Obtener la frecuencia actual del FRO.
- void hal_syscon_external_crystal_config (uint32_t crystal_freq)
  Configurar el ext clock a partir de un cristal externo.
- void hal_syscon_external_clock_config (uint32_t external_clock_freq)
  Configurar el ext clock a partir de una fuente de clock externa.
- void hal_syscon_fro_clock_config (uint8_t direct)
  Configurar el clock FRO.
- void hal_syscon_fro_clock_disable (void)
  Inhabilitar el FRO.
- void hal_syscon_clkout_config (hal_gpio_portpin_en portpin, 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_frg_config (uint8_t inst, hal_syscon_frg_clock_sel_en clock_source, uint32_t mul)
  Configurar el divisor fraccional.
- void hal_syscon_watchdog_oscillator_config (hal_syscon_watchdog_clkana_sel_en clkana_sel, uint8_t div)
  Configuración del watchdog oscillator.
- uint32_t hal_syscon_peripheral_clock_get (hal_syscon_peripheral_sel_en peripheral)
  Obtener la frecuencia de clock en Hz configurada para cierto periférico.
- void hal_syscon_iocon_glitch_divider_set (hal_syscon_iocon_glitch_sel_en sel, uint32_t div)
  Configurar divisor para el clock de glitches del IOCON.
- void hal_syscon_pll_clock_config (hal_syscon_pll_source_sel_en clock_source, uint32_t freq)
  Configurar el PLL.
- uint32_t hal_syscon_pll_clock_get (void)
  Obtener frecuencia actual configurada del PLL.

2.7.2. Documentación de las enumeraciones

2.7.2.1. hal_syscon_system_clock_sel_en

`enum hal_syscon_system_clock_sel_en`

Selección de fuente de clock para el clock principal

Valores de enumeraciones

HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO	Free Running Oscillator
HAL_SYSCON_SYSTEM_CLOCK_SEL_EXT	Clcok externo
HAL_SYSCON_SYSTEM_CLOCK_SEL_WATCHDOG	Watchdog Oscillator
HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO_DIV	Free Running Oscillator dividido 2
HAL_SYSCON_SYSTEM_CLOCK_SEL_PLL	Phase Locked Loop

2.7.2.2. hal_syscon_clkout_source_sel_en

`enum hal_syscon_clkout_source_sel_en`

Selección de fuente de clock para la salida CLKOUT

Valores de enumeraciones

HAL_SYSCON_CLKOUT_SOURCE_SEL_FRO	Free Running Oscillator
HAL_SYSCON_CLKOUT_SOURCE_SEL_MAIN_CLOCK	Clock principal
HAL_SYSCON_CLKOUT_SOURCE_SEL_SYS_PLL	Phase Locked Loop
HAL_SYSCON_CLKOUT_SOURCE_SEL_EXT_CLOCK	Clock externo
HAL_SYSCON_CLKOUT_SOURCE_SEL_WATCHDOG_OSC	Watchdog Oscillator

2.7.2.3. hal_syscon_frg_clock_sel_en

`enum hal_syscon_frg_clock_sel_en`

Valores de enumeraciones

HAL_SYSCON_FRG_CLOCK_SEL_FRO	Free Running Oscillator
HAL_SYSCON_FRG_CLOCK_SEL_MAIN_CLOCK	Clock principal
HAL_SYSCON_FRG_CLOCK_SEL_SYS_PLL	Phase Locked Loop
HAL_SYSCON_FRG_CLOCK_SEL_NONE	Ninguno

2.7.2.4. hal_syscon_watchdog_clkana_sel_en

`enum hal_syscon_watchdog_clkana_sel_en`

Selección de frecuencia base para el watchdog oscillator

Valores de enumeraciones

HAL_SYSCON_WATCHDOG_CLKANA_0KHZ	0MHz
HAL_SYSCON_WATCHDOG_CLKANA_600KHZ	0.6MHz
HAL_SYSCON_WATCHDOG_CLKANA_1050KHZ	1.05MHz
HAL_SYSCON_WATCHDOG_CLKANA_1400KHZ	1.4MHz
HAL_SYSCON_WATCHDOG_CLKANA_1750KHZ	1.75MHz
HAL_SYSCON_WATCHDOG_CLKANA_2100KHZ	2.1MHz
HAL_SYSCON_WATCHDOG_CLKANA_2400KHZ	2.4MHz
HAL_SYSCON_WATCHDOG_CLKANA_3000KHZ	3MHz
HAL_SYSCON_WATCHDOG_CLKANA_3250KHZ	3.25MHz
HAL_SYSCON_WATCHDOG_CLKANA_3500KHZ	3.5MHz
HAL_SYSCON_WATCHDOG_CLKANA_3750KHZ	3.75MHz
HAL_SYSCON_WATCHDOG_CLKANA_4000KHZ	4MHz
HAL_SYSCON_WATCHDOG_CLKANA_4200KHZ	4.2MHz
HAL_SYSCON_WATCHDOG_CLKANA_4400KHZ	4.4MHz
HAL_SYSCON_WATCHDOG_CLKANA_4600KHZ	4.6MHz

2.7.2.5. hal_syscon_peripheral_sel_en

`enum hal_syscon_peripheral_sel_en`

Selección de periférico para controlar fuente de clock

Valores de enumeraciones

HAL_SYSCON_PERIPHERAL_SEL_UART0	UART0
HAL_SYSCON_PERIPHERAL_SEL_UART1	UART1
HAL_SYSCON_PERIPHERAL_SEL_UART2	UART2
HAL_SYSCON_PERIPHERAL_SEL_UART3	UART3
HAL_SYSCON_PERIPHERAL_SEL_UART4	UART4
HAL_SYSCON_PERIPHERAL_SEL_IIC0	IIC0
HAL_SYSCON_PERIPHERAL_SEL_IIC1	IIC1
HAL_SYSCON_PERIPHERAL_SEL_IIC2	IIC2
HAL_SYSCON_PERIPHERAL_SEL_IIC3	IIC3
HAL_SYSCON_PERIPHERAL_SEL_SPI0	SPI0
HAL_SYSCON_PERIPHERAL_SEL_SPI1	SPI1

2.7.2.6. hal_syscon_peripheral_clock_sel_en

`enum hal_syscon_peripheral_clock_sel_en`

Selección de fuente de clock para los periféricos

Valores de enumeraciones

HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRO	Free Running Oscillator
HAL_SYSCON_PERIPHERAL_CLOCK_SEL_MAIN	Clock principal
HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRG0	Generador fraccional 0
HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRG1	Generador fraccional 1
HAL_SYSCON_PERIPHERAL_CLOCK_SEL_FRO_DIV	Free Running Oscillator dividido por 2
HAL_SYSCON_PERIPHERAL_CLOCK_SEL_NONE	Ninguna

2.7.2.7. hal_syscon_iocon_glitch_sel_en

`enum hal_syscon_iocon_glitch_sel_en`

Selección de banco de división para filtros de glitches

Valores de enumeraciones

HAL_SYSCON_IOCON_GLITCH_SEL↩_0	Banco 0
HAL_SYSCON_IOCON_GLITCH_SEL↩_1	Banco 1
HAL_SYSCON_IOCON_GLITCH_SEL↩_2	Banco 2
HAL_SYSCON_IOCON_GLITCH_SEL↩_3	Banco 3
HAL_SYSCON_IOCON_GLITCH_SEL↩_4	Banco 4
HAL_SYSCON_IOCON_GLITCH_SEL↩_5	Banco 5
HAL_SYSCON_IOCON_GLITCH_SEL↩_6	Banco 6
HAL_SYSCON_IOCON_GLITCH_SEL↩_7	Banco 7

2.7.2.8. hal_syscon_pll_source_sel_en

`enum hal_syscon_pll_source_sel_en`

Fuente de clock para el PLL

Valores de enumeraciones

HAL_SYSCON_PLL_SOURCE_SEL_FRO	Free Running Oscillator
HAL_SYSCON_PLL_SOURCE_SEL_EXT_CLK	Clock externo
HAL_SYSCON_PLL_SOURCE_SEL_WATCHDOG	Watchdog Oscillator
HAL_SYSCON_PLL_SOURCE_SEL_FRO_DIV	Free Running Oscillator dividido por 2

2.7.3. Documentación de las funciones

2.7.3.1. hal_syscon_system_clock_get()

```
uint32_t hal_syscon_system_clock_get (
            void  )
```

Obtener la frecuencia actual del main clock.

Devuelve

Frecuencia del main clock en Hz

2.7.3.2. hal_syscon_system_clock_set_source()

```
void hal_syscon_system_clock_set_source (
            hal_syscon_system_clock_sel_en clock_source )
```

Configuración de fuente de clock para el clock principal.

Parámetros

in	*clock_source*	Selección deseada

Ejemplos

Ejemplo_SYSCON.c.

2.7.3.3. hal_syscon_system_clock_set_divider()

```
void hal_syscon_system_clock_set_divider (
            uint8_t div )
```

Fijar el divisor del clock principal.

Parámetros

in	*div*	Divisor deseado. Cero inhabilita el clock principal

2.7.3.4. hal_syscon_fro_clock_get()

```
uint32_t hal_syscon_fro_clock_get (
            void  )
```

Obtener la frecuencia actual del FRO.

Devuelve

Frecuencia del FRO en Hz

2.7.3.5. hal_syscon_external_crystal_config()

```
void hal_syscon_external_crystal_config (
            uint32_t crystal_freq )
```

Configurar el ext clock a partir de un cristal externo.

Parámetros

in	*crystal_freq*	Frecuencia del cristal externo utilizado

2.7.3.6. hal_syscon_external_clock_config()

```
void hal_syscon_external_clock_config (
            uint32_t external_clock_freq )
```

Configurar el ext clock a partir de una fuente de clock externa.

Parámetros

in	*external_clock_freq*	Frecuencia de la fuente de clock externa en Hz

2.7.3.7. hal_syscon_fro_clock_config()

```
void hal_syscon_fro_clock_config (
            uint8_t direct )
```

Configurar el clock FRO.

Nota

Esta función habilita el FRO

Parámetros

in	*direct*	Si es distinto de cero se omite el divisor del FRO

Ejemplos

Ejemplo_SYSCON.c.

2.7.3.8. hal_syscon_fro_clock_disable()

```
void hal_syscon_fro_clock_disable (
            void  )
```

Inhabilitar el FRO.

2.7.3.9. hal_syscon_clkout_config()

```
void hal_syscon_clkout_config (
            hal_gpio_portpin_en portpin,
            hal_syscon_clkout_source_sel_en clock_source,
            uint8_t divider )
```

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

Parámetros

in	*portpin*	Número de puerto/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

Ejemplos

Ejemplo_SYSCON.c.

2.7.3.10. hal_syscon_frg_config()

```
void hal_syscon_frg_config (
            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

2.7.3.11. hal_syscon_watchdog_oscillator_config()

```
void hal_syscon_watchdog_oscillator_config (
            hal_syscon_watchdog_clkana_sel_en clkana_sel,
            uint8_t div )
```

Configuración del watchdog oscillator.

Parámetros

in	*clkana_sel*	Selección de frecuencia base del oscilador
in	*div*	Divisor. El valor efectivo de división es: 2 (1 + div)

Nota

Recordar que la presición de este oscilador es de más/menos 40 %

Parámetros

in	*clkana_sel*	Selección de frecuencia base del oscilador
in	*div*	Divisor

2.7.3.12. hal_syscon_peripheral_clock_get()

```
uint32_t hal_syscon_peripheral_clock_get (
            hal_syscon_peripheral_sel_en peripheral )
```

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

Parámetros

in	*peripheral*	Periférico deseado

Devuelve

Frecuencia en Hz del clock del periférico

2.7.3.13. hal_syscon_iocon_glitch_divider_set()

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

Configurar divisor para el clock de glitches del IOCON.

Parámetros

in	*sel*	Selección de divisor
in	*div*	Valor de división deseado

2.7.3.14. hal_syscon_pll_clock_config()

```
void hal_syscon_pll_clock_config (
            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

2.7.3.15. hal_syscon_pll_clock_get()

```
uint32_t hal_syscon_pll_clock_get (
            void  )
```

Obtener frecuencia actual configurada del PLL.

Devuelve

Frecuencia actual del PLL en Hz

2.8. Tick del Sistema (SYSTICK)

2.8.1. Descripción detallada

Introducción

El periférico *SYSTICK* es un timer básico cuya funcionalidad principal es mantener una base de tiempos, en general fija, para múltiples tareas que necesiten cierta periodicidad o una base de tiempo. La frecuencia con la que ocurren dichos ticks es configurable y valores típicos pueden ser: 1 milisegundo, 2.5 milisegundos, 10 milisegundos entre otros.

Fuente de clock

El periférico admite dos ocpiones de fuente de clock para el mismo:

- Clock principal del sistema
- Clock principal del sistema dividido 2

Configuración de tiempo de tick

Para configurar el tiempo de tick del periférico, el mismo cuenta con un contador descendente de *24 bits* el cual cuando desborda, activa un flag, pudiendo el mismo generar una interrupción o no. La librería implementa una función hal_systick_init a la cual simplemente se le asigna el tiempo de tick deseado en microsegundos y automáticamente configura el periférico para generar ticks con dicha base de tiempo, pudiendo ejecutar un callback en caso de ser necesario.

Callback de tick

Usualmente el periférico se configura para generar interrupciones y ejecutar un callback asociado en dicho momento. Es importante recordar que dicho callback se ejecutará en el contexto de una interrupción, por lo que el callback a ejecutar tendrá que tener todas las consideraciones pertinentes. En la librería, el callback puede ser configurado al llamar a la función hal_systick_init o luego de haber inicializado el periférico, si es necesario cambiar el callback, o inhabilitarlo, se puede llamar a la función hal_systick_update_callback.

Inhibición de tick

Para ciertos procesos críticos, es deseable inhibir las interrupciones del *SYSTICK*. Para este propósito se disponen de las funciones hal_systick_inhibit_set y hal_systick_inhibit_clear.

typedefs

- typedef void(∗ hal_systick_callback_t) (void)

Funciones

- void hal_systick_init (uint32_t tick_us, void(∗callback)(void))

  Inicializacion del SYSTICK.
- void hal_systick_update_callback (hal_systick_callback_t callback)

  Actualizar callback del SYSTICK.
- void hal_systick_inhibit_set (void)

  Inhabilitar interrupciones de SYSTICK.
- void hal_systick_inhibit_clear (void)

  Habilitar interrupciones de SYSTICK.

2.8.2. Documentación de los 'typedefs'

2.8.2.1. hal_systick_callback_t

```
typedef void(* hal_systick_callback_t) (void)
```

Tipo de dato para callbacks del *SYSTICK*

2.8.3. Documentación de las funciones

2.8.3.1. 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, Ejemplo_DAC.c y Ejemplo_GPIO.c.

2.8.3.2. 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

Si se pasa como parámetro *NULL*, se inhabilitarán las interrupciones

Parámetros

in	*callback*	Nuevo callback a ejecutar en cada tick

2.8.3.3. hal_systick_inhibit_set()

```
void hal_systick_inhibit_set (
            void  )
```

Inhabilitar interrupciones de *SYSTICK*.

2.8.3.4. hal_systick_inhibit_clear()

```
void hal_systick_inhibit_clear (
            void  )
```

Habilitar interrupciones de *SYSTICK*.

2.9. Wake Up Timer (WKT)

2.9.1. Descripción detallada

Introducción

Este periférico es un contador básico con posibilidad de tres fuentes distintas de clock. El mismo puede ser utilizado para despertar al microcontrolador de modos de bajo consumo siempre y cuando la fuente de clock sea correctamente seleccionada. También se puede utilizar como un timer genérico.

Al cargar un valor distinto de cero en la cuenta del periférico, el mismo se enciende automáticamente y comienza el conteo. Una vez transcurrido el conteo, el periférico genera la interrupción correspondiente y se vuelve a apagar hasta un próximo conteo, es decir, es un timer de tipo *one-shot*.

Nota

El periférico se inicializa mediante la función hal_wkt_init y se dispara un conteo mediante las funciones hal_wkt_start_count o hal_wkt_start_count_with_value dependiendo de la necesidad del usuario.

Fuentes de clock

Se puede excitar al periférico con tres fuentes de clock distintas:

- *FRO dividido*: FRO dividido por 16
- *Oscilador de bajo consumo*: Oscilador siempre encendido con una frecuencia de 10KHz más/menos 40 %. La ventaja de este oscilador es que sigue encendido inclusive en modos de ultra bajo consumo, pudiendo despertar al microcontrolador de los mismos.
- *Fuente de clock externo en el pin WKTLCKIN: Ubicado en Puerto 0 ; Pin 28*

Nota

La fuente de clock es seleccionada mediante la función hal_wkt_select_clock_source o en la inicialización mediante la función hal_wkt_init.

Nota acerca de la carga del conteo

En la librería se implementan dos funciones para generar la carga del contador del periférico, las mismas son hal_wkt_start_count y hal_wkt_start_count_with_value. La diferencia entre una y otra, es que en la primera, la librería pide como argumento un tiempo en microsegundos y hace la cuenta necesaria para aproximarse lo más posible y cargar el conteo adecuado, mientras que la segunda directamente pide el valor a cargar al usuario. Cabe destacar que la primer función, utiliza números con punto flotante, por lo que su ejecución tarda un tiempo considerable, dado que el microcontrolador no tiene unidad de punto flotante de hardware incorporada. El criterio para utilizar una función u otra es:

- *Si los tiempos de conteo son lo suficientemente largos, y/o la presición no es un factor demasiado importante (por ejemplo, para salir de un modo de bajo consumo tal vez no sea necesaria una presición demasiado alta) se puede utilizar directametne la función hal_wkt_start_count.*
- *Si el tiempo de conteo es bajo, o la presición es un factor vital en el conteo (por ejemplo, se utiliza el periférico para un delay de corto tiempo preciso) es recomendable utilizar la función hal_wkt_start_count_with_value.*

typedefs

- typedef void(∗ hal_wkt_callback_t) (void)

  Tipo de dato para el callback de interrupción del WKT.

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, hal_wkt_callback_t callback)

  Inicializar el WKT.
- void hal_wkt_select_clock_source (hal_wkt_clock_source_en clock_sel, uint32_t ext_clock_value)

  Configurar fuente de clock para el WKT.
- void hal_wkt_register_callback (hal_wkt_callback_t new_callback)

  Registrar un callback para la interrupción del WKT.
- void hal_wkt_start_count (uint32_t time_useg)

  Iniciar el conteo con el WKT en base a un tiempo.
- void hal_wkt_start_count_with_value (uint32_t value)

  Iniciar el conteo con el WKT en base a un valor.

2.9.2. Documentación de los 'typedefs'

2.9.2.1. hal_wkt_callback_t

```
typedef void(* hal_wkt_callback_t) (void)
```

Tipo de dato para el callback de interrupción del *WKT*.

Nota

Es importante recordar que estos callbacks se ejecutan en el contexto de una interrupción, por lo que el usuario deberá tener en cuenta todas las consideraciones necesarias a la hora de escribir el mismo.

2.9.3. Documentación de las enumeraciones

2.9.3.1. hal_wkt_clock_source_en

```
enum hal_wkt_clock_source_en
```

Selección de fuente de clock para el *WKT*

Valores de enumeraciones

HAL_WKT_CLOCK_SOURCE_FRO_DIV	Fuente de clock FRO dividido
HAL_WKT_CLOCK_SOURCE_LOW_POWER_OSC	Fuente de clock oscilador de bajo consumo
HAL_WKT_CLOCK_SOURCE_EXTERNAL	Fuente de clock externa

2.9.4. Documentación de las funciones

2.9.4.1. 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*	Selección de clock deseada para el WKT
in	*ext_clock_value*	Valor de clock externo (si la selección es interna, no importa este parámetro)
in	*callback*	Callback a ejecutar en la interrupción del WKT

Nota

Es importante recordar que estos callbacks se ejecutan en el contexto de una interrupción, por lo que el usuario deberá tener en cuenta todas las consideraciones necesarias a la hora de escribir el mismo.

Parámetros

in	*clock_sel*	Selección de clock deseada para el WKT
in	*ext_clock_value*	Valor de clock externo (si la selección es interna, no importa este parámetro)
in	*callback*	Callback a ejecutar en la interrupción del WKT

Ejemplos

Ejemplo_PININT.c y Ejemplo_WKT.c.

2.9.4.2. hal_wkt_select_clock_source()

```
void hal_wkt_select_clock_source (
            hal_wkt_clock_source_en clock_sel,
            uint32_t ext_clock_value )
```

Configurar fuente de clock para el WKT.

Parámetros

in	*clock_sel*	Selección de clock deseada para el WKT
in	*ext_clock_value*	Valor de clock externo (si la selección es interna, no importa este parámetro)

2.9.4.3. hal_wkt_register_callback()

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

Registrar un callback para la interrupción del WKT.

Parámetros

in	*new_callback*	Nuevo callback para la interrupción del WKT

Nota

Es importante recordar que estos callbacks se ejecutan en el contexto de una interrupción, por lo que el usuario deberá tener en cuenta todas las consideraciones necesarias a la hora de escribir el mismo.

Parámetros

in	*new_callback*	Nuevo callback para la interrupción del WKT

2.9.4.4. hal_wkt_start_count()

```
void hal_wkt_start_count (
             uint32_t time_useg )
```

Iniciar el conteo con el WKT en base a un tiempo.

Nota

Esta función utiliza variables con punto flotante por lo que tarda un tiempo considerable en ejecutarse.

Parámetros

in	*time_useg*	Tiempo en microsegundos deseado (se redondeará al valor primer posible hacia arriba)
in	*time_useg*	Tiempo en microsegundos deseado (se redondeará al valor primer posible hacia arriba)

Ejemplos

Ejemplo_PININT.c.

2.9.4.5. hal_wkt_start_count_with_value()

```
void hal_wkt_start_count_with_value (
             uint32_t value )
```

Iniciar el conteo con el WKT en base a un valor.

Nota

El usuario es responsable de colocar un valor que tenga sentido en base al clock utilizado.

Parámetros

in	*value*	Valor deseado a poner en el conteo (útil para una actualización mas rapida)

El usuario es responsable de colocar un valor que tenga sentido en base al clock utilizado.

Parámetros

in	*value*	Valor deseado a poner en el conteo (útil para una actualización mas rapida)

Ejemplos

Ejemplo_WKT.c.

Capítulo 3

Documentación de las estructuras de datos

3.1. 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

3.2. 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)

3.2.1. Documentación de los campos

3.2.1.1. duty

```
uint32_t hal_ctimer_pwm_channel_config_t::duty
```

Duty en decimas de porciento (1 equivale a 0.1 %)

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

- includes/hal/HAL_CTIMER.h

3.3. 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)

3.3.1. Documentación de los campos

3.3.1.1. clock_div

```
uint32_t hal_ctimer_pwm_config_t::clock_div
```

Corresponde al numero deseado a dividir menos 1.

3.3.1.2. pwm_period_useg

```
uint32_t hal_ctimer_pwm_config_t::pwm_period_useg
```

Periodo del PWM en microsegundos.

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

- includes/hal/HAL_CTIMER.h

3.4. 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

3.5. 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 4

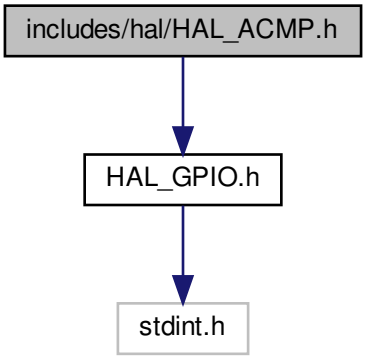
Documentación de archivos

4.1. Referencia del Archivo includes/hal/HAL_ACMP.h

Declaraciones a nivel de aplicación del periférico ADC (LPC845)

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

Dependencia gráfica adjunta para HAL_ACMP.h:



Estructuras de datos

- struct hal_acpm_config_t
- struct hal_acmp_ladder_config_t

Enumeraciones

- enum hal_acmp_output_control_en {
  HAL_ACMP_OUTPUT_DIRECT = 0,
  HAL_ACMP_OUTPUT_SYNC }
- enum hal_acmp_hysteresis_sel_en {
  HAL_ACMP_HYSTERESIS_NONE = 0,
  HAL_ACMP_HYSTERESIS_5mV,
  HAL_ACMP_HYSTERESIS_10mV,
  HAL_ACMP_HYSTERESIS_20mV }
- enum hal_acmp_ladder_vref_sel_en {
  HAL_ACMP_LADDER_VREF_VDD = 0,
  HAL_ACMP_LADDER_VREF_VDDCMP_PIN }
- enum hal_acmp_edge_sel_en {
  HAL_ACMP_EDGE_FALLING = 0,
  HAL_ACMP_EDGE_RISING,
  HAL_ACMP_EDGE_BOTH }
- enum hal_acmp_input_voltage_sel_en {
  HAL_ACMP_INPUT_VOLTAGE_VLADDER_OUT = 0,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I1,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I2,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I3,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I4,
  HAL_ACMP_INPUT_VOLTAGE_ACMP_I5,
  HAL_ACMP_INPUT_VOLTAGE_BANDGAP,
  HAL_ACMP_INPUT_VOLTAGE_DACOUT0 }

Funciones

- void hal_acmp_init (void)

  Inicialización del periférico Comparador Analógico.
- void hal_acmp_deinit (void)

  De-inicialización del periferico Comparador Analógico.
- void hal_acmp_config (const hal_acpm_config_t *acmp_config)

  Configuración de parámetros generales del comparador analógico.
- void hal_acmp_ladder_config (const hal_acmp_ladder_config_t *config)

  Configuración de la Voltage Ladder del comparador analógico.
- void hal_acmp_input_select (hal_acmp_input_voltage_sel_en positive_input, hal_acmp_input_voltage_sel_en negative_input)

  Selecciona las entradas positiva y negativa deseadas para el comparador.
- void hal_acmp_output_pin_set (hal_gpio_portpin_en port_pin)

  Asigna la salida del comparador a un pin externo del microcontrolador.
- void hal_acmp_output_pin_clear ()

  Deshace la asignación de la salida del comparador a un pin externo del microcontrolador.

4.1.1. Descripción detallada

Declaraciones a nivel de aplicación del periférico ADC (LPC845)

Autor

Esteban E. Chiama

Fecha

4/2020

Versión

1.0

4.2. Referencia del Archivo includes/hal/HAL_ADC.h

Declaraciones a nivel de aplicación del periférico ADC (LPC845)

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

Dependencia gráfica adjunta para HAL_ADC.h:

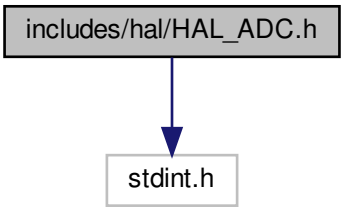
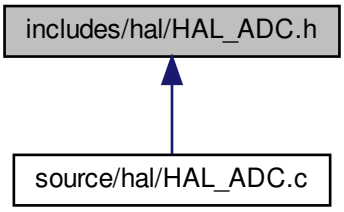


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct hal_adc_sequence_config_t
- struct hal_adc_sequence_result_t
- struct hal_adc_channel_compare_result_t

typedefs

- typedef void(∗ adc_sequence_interrupt_t) (void)

  Tipo de dato para callback de interrupcion de sequencia.
- typedef void(∗ adc_comparison_interrupt_t) (void)

  Tipo de dato para callback de interrupcion de comparación.

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 }
- enum hal_adc_threshold_sel_en {
  HAL_ADC_THRESHOLD_SEL_0 = 0,
  HAL_ADC_THRESHOLD_SEL_1 }

- enum hal_adc_threshold_interrupt_sel_en {
  HAL_ADC_THRESHOLD_IRQ_SEL_DISABLED = 0,
  HAL_ADC_THRESHOLD_IRQ_SEL_OUTSIDE,
  HAL_ADC_THRESHOLD_IRQ_SEL_CROSSING }
- enum hal_adc_compare_range_result_en {
  HAL_ADC_COMPARISON_RANGE_INSIDE = 0,
  HAL_ADC_COMPARISON_RANGE_BELOW,
  HAL_ADC_COMPARISON_RANGE_ABOVE }
- enum hal_adc_compare_crossing_result_en {
  HAL_ADC_COMPARISON_NO_CROSSING = 0,
  HAL_ADC_COMPARISON_CROSSING_DOWNWARD = 2,
  HAL_ADC_COMPARISON_CROSSING_UPWARD }

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_sequence_config (hal_adc_sequence_sel_en sequence, const hal_adc_sequence_config_t *config)

  Configurar una secuencia de conversión.
- void hal_adc_sequence_start (hal_adc_sequence_sel_en sequence)

  Disparar conversiones en una secuencia.
- void hal_adc_sequence_stop (hal_adc_sequence_sel_en sequence)

  Detener conversiones en una secuencia de conversión.
- hal_adc_sequence_result_en hal_adc_sequence_get_result (hal_adc_sequence_sel_en sequence, hal_adc_sequence_result_t *result)

  Obtener resultado de la secuencia.
- void hal_adc_threshold_config (hal_adc_threshold_sel_en threshold, uint16_t low, uint16_t high)

  Configurar valor de umbral de comparación.
- void hal_adc_threshold_channel_config (uint8_t adc_channel, hal_adc_threshold_sel_en threshold, hal_adc_threshold_interrupt_sel_en irq_mode)

  Configura un canal para utilizar la funcionalidad de comparación con un umbral y su tipo de interrupción deseada.
- void hal_adc_threshold_register_interrupt (adc_comparison_interrupt_t callback)

  Registrar un callabck de interrupción para interrupción por threshold.
- void hal_adc_threshold_get_comparison_results (hal_adc_channel_compare_result_t *results)

  Obtener resultados de comparación de la última conversión.

4.2.1. Descripción detallada

Declaraciones a nivel de aplicación del periférico ADC (LPC845)

Autor

Augusto Santini

Esteban E. Chiama

Fecha

3/2020

Versión

1.0

4.3. 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:

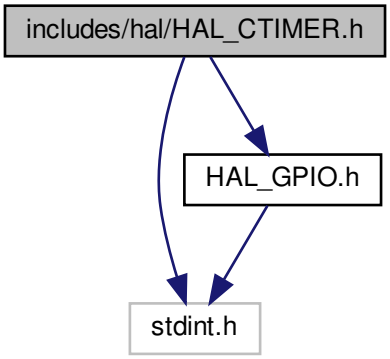
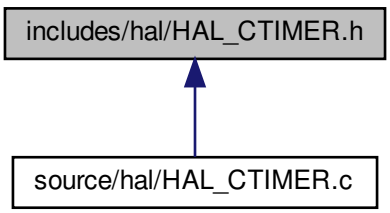


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



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_NOTHING = 0,
  HAL_CTIMER_MATCH_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_CTIMER_MATCH_2,
  HAL_CTIMER_MATCH_3 }
- enum **hal_ctimer_pwm_channel_sel_en** {
  HAL_CTIMER_PWM_CHANNEL_0 = 0,
  HAL_CTIMER_PWM_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_match_config (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_timer_mode_match_change_value (hal_ctimer_match_sel_en match, uint32_t match_↩ value_useg)

  Cambia el valor de MATCH del CTIMER seleccionado.
- uint8_t hal_ctimer_match_read_output (hal_ctimer_match_sel_en match)

  Leer estado de match externo.
- void hal_ctimer_match_set_output (hal_ctimer_match_sel_en match)

  Pone la señal de salida EM# (External Match #) en 1.
- void hal_ctimer_match_clear_output (hal_ctimer_match_sel_en match)

  Pone la señal de salida EMn (External Match n) en 0.
- void hal_ctimer_pwm_mode_init (const hal_ctimer_pwm_config_t ∗config)

  Inicializar el CTIMER en modo PWM.
- void hal_ctimer_pwm_mode_period_set (uint32_t period_useg)

  Actualizar el periodo en modo PWM.
- void hal_ctimer_pwm_mode_channel_config (hal_ctimer_pwm_channel_sel_en channel_sel, const hal_ctimer_pwm_channel_config_t ∗channel_config)

  Actualizar configuracion de algun canal de PWM.

4.3.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico CTIMER (LPC845)

Autor

Augusto Santini

Esteban E. Chiama

Fecha

3/2020

Versión

1.0

4.3.2. Documentación de las funciones

4.3.2.1. 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)

4.3.2.2. hal_ctimer_timer_mode_match_config()

```
void hal_ctimer_timer_mode_match_config (
            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

4.3.2.3. hal_ctimer_timer_mode_run()

```
void hal_ctimer_timer_mode_run (
            void  )
```

Habilitar el conteo del ctimer.

4.3.2.4. hal_ctimer_timer_mode_stop()

```
void hal_ctimer_timer_mode_stop (
            void  )
```

Inhabilitar el conteo del ctimer.

4.3.2.5. hal_ctimer_timer_mode_reset()

```
void hal_ctimer_timer_mode_reset (
            void  )
```

Reiniciar el conteo del ctimer.

4.3.2.6. hal_ctimer_timer_mode_match_change_value()

```
void hal_ctimer_timer_mode_match_change_value (
            hal_ctimer_match_sel_en match,
            uint32_t match_value_useg )
```

Cambia el valor de MATCH del CTIMER seleccionado.

Si el match deseado está configurado para realizar *reload on match*, se escribe el nuevo valor de match será una actualización efectiva en cuanto el conteo actual alcance dicho match. Caso contrario, la actualización del valor de match es inmediata.

Parámetros

in	*match_sel*	Match a configurar
in	*match_value_useg*	Nuevo valor de match, en useg, deseado.

4.3.2.7. hal_ctimer_match_read_output()

```
uint8_t hal_ctimer_match_read_output (
            hal_ctimer_match_sel_en match )
```

Leer estado de match externo.

Parámetros

in	*match*	Numero de match externo a consultar

Devuelve

Estado del match actual

4.3.2.8. hal_ctimer_match_set_output()

```
void hal_ctimer_match_set_output (
            hal_ctimer_match_sel_en match )
```

Pone la señal de salida EM# (External Match #) en 1.

Parámetros

in	*match*	Numero de match externo a configurar

4.3.2.9. hal_ctimer_match_clear_output()

```
void hal_ctimer_match_clear_output (
            hal_ctimer_match_sel_en match )
```

Pone la señal de salida EMn (External Match n) en 0.

Parámetros

in	*match*	Numero de match externo a configurar

Pone la señal de salida EMn (External Match n) en 0.

Parámetros

in	*match*	Numero de match externo a configurar

4.3.2.10. 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

4.3.2.11. hal_ctimer_pwm_mode_period_set()

```
void hal_ctimer_pwm_mode_period_set (
            uint32_t period_useg )
```

Actualizar el periodo en modo PWM.

Parámetros

in	*period_useg*	Nuevo periodo deseado en microsegundos

4.3.2.12. hal_ctimer_pwm_mode_channel_config()

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

Actualizar configuracion de algun canal de PWM.

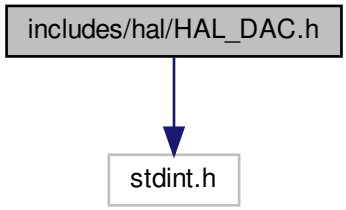
Parámetros

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

4.4. 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)
  Inicialización del DAC.
- void hal_dac_update_value (hal_dac_en dac, uint16_t new_value)
  Actualización del valor actual del DAC.
- void hal_dac_config_ctrl (hal_dac_en dac, hal_dac_ctrl_config_t ∗config)
  Configuración del registro de control del DAC.

4.4.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico DAC (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.5. 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:

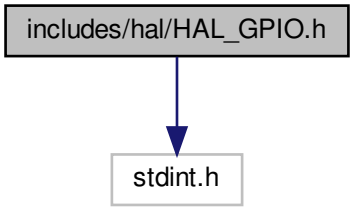
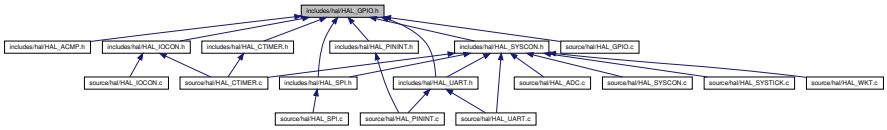


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



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_PORTPIN_0_27,
HAL_GPIO_PORTPIN_0_28,
HAL_GPIO_PORTPIN_0_29,
HAL_GPIO_PORTPIN_0_30,
HAL_GPIO_PORTPIN_0_31,
HAL_GPIO_PORTPIN_1_0,
HAL_GPIO_PORTPIN_1_1,
HAL_GPIO_PORTPIN_1_2,
HAL_GPIO_PORTPIN_1_3,
HAL_GPIO_PORTPIN_1_4,
HAL_GPIO_PORTPIN_1_5,
HAL_GPIO_PORTPIN_1_6,
HAL_GPIO_PORTPIN_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 dirección de una GPIO.
- void hal_gpio_set_pin (hal_gpio_portpin_en portpin)

  Fijar estado de una GPIO (sin importar máscara)
- void hal_gpio_set_port (hal_gpio_port_en port, uint32_t bits_to_set)

  Fijar estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_set_port (hal_gpio_port_en port, uint32_t bits_to_set)

  Fijar estado de pines de un puerto (teniendo en cuenta máscara)
- void hal_gpio_clear_pin (hal_gpio_portpin_en portpin)

  Limpiar estado de una GPIO (sin importar máscara)
- void hal_gpio_clear_port (hal_gpio_port_en port, uint32_t bits_to_clear)

  Limpiar estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_clear_port (hal_gpio_port_en port, uint32_t bits_to_clear)

  Limpiar estado de pines de un puerto (teniendo en cuenta máscara)

- void hal_gpio_toggle_pin (hal_gpio_portpin_en portpin)

  Invertir estado de una GPIO (sin importar máscara)
- void hal_gpio_toggle_port (hal_gpio_port_en port, uint32_t bits_to_toggle)

  Invertir estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_toggle_port (hal_gpio_port_en port, uint32_t bits_to_toggle)

  Invertir estado de pines de un puerto (teniendo en cuenta máscara)
- uint8_t hal_gpio_read_pin (hal_gpio_portpin_en portpin)

  Leer el estado de una GPIO (sin importar máscara)
- uint32_t hal_gpio_read_port (hal_gpio_port_en port)

  Leer estado de un puerto (sin importar máscara)
- uint32_t hal_gpio_masked_read_port (hal_gpio_port_en port)

  Leer estado de un puerto (teniendo en cuenta máscara)
- void hal_gpio_set_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Fijar enmascaramiento de pines en un puerto.
- void hal_gpio_clear_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Limpiar enmascaramiento de pines en un puerto.
- void hal_gpio_toggle_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Invertir enmascaramiento de pines en un puerto.

4.5.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico GPIO (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.6. Referencia del Archivo includes/hal/HAL_IOCON.h

Declaraciones a nivel de aplicacion del periferico IOCON (LPC845)

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

Dependencia gráfica adjunta para HAL_IOCON.h:

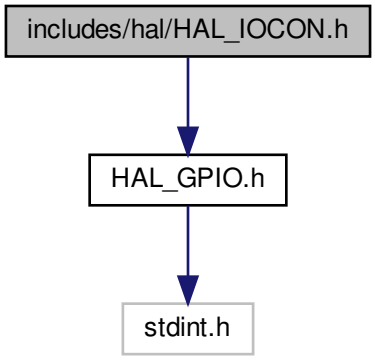
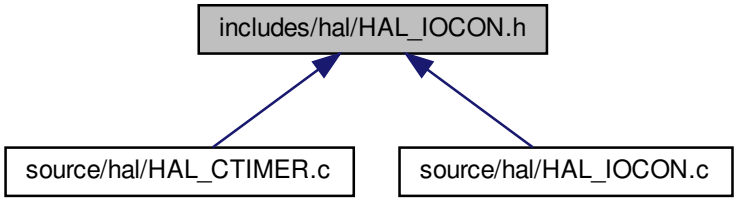


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



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.

4.6.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico IOCON (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.7. 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:

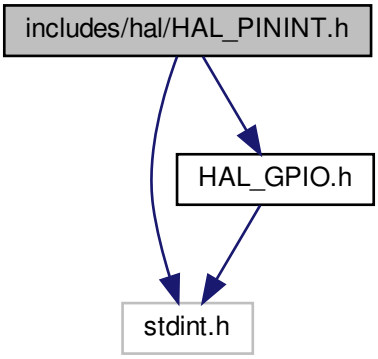
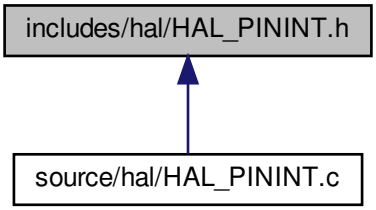


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



typedefs

- typedef void(∗ hal_pinint_callback_t) (void)

  Tipo de dato para callback de PININT.

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_edge_detections_en {
  HAL_PININT_EDGE_DETECTIONS_NONE = 0,
  HAL_PININT_EDGE_DETECTIONS_RISING,
  HAL_PININT_EDGE_DETECTIONS_FALLING,
  HAL_PININT_EDGE_DETECTIONS_BOTH }
- enum hal_pinint_level_detections_en {
  HAL_PININT_LEVEL_DETECTIONS_NONE = 0,
  HAL_PININT_LEVEL_DETECTIONS_HIGH,
  HAL_PININT_LEVEL_DETECTIONS_LOW }

Funciones

- void hal_pinint_init (void)

  Inicialización del periférico.
- void hal_pinint_deinit (void)

  De-Inicialización del periférico.
- void hal_pinint_channel_config (hal_pinint_channel_en channel, hal_gpio_portpin_en portpin, hal_pinint_callback_t callback)

  Configuración de canal de PININT.
- void hal_pinint_edge_detections_config (hal_pinint_channel_en channel, hal_pinint_edge_detections_en edge)

  Configurar detecciones por flanco.
- void hal_pinint_level_detections_config (hal_pinint_channel_en channel, hal_pinint_level_detections_en level)

  Configurar detecciones por nivel.

4.7.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico PININT (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.8. 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:

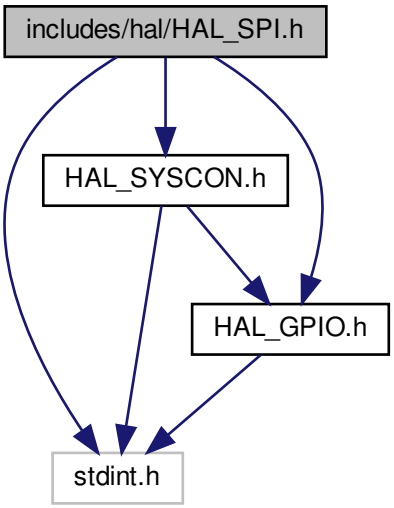
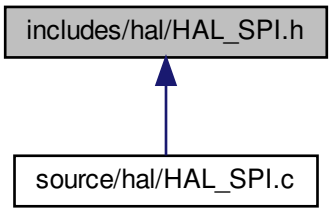


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



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_tx_config (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_tx_register_callback (hal_spi_sel_en inst, void(*new_callback)(void))
  Actualizar callback en TXRDY.
- void hal_spi_master_mode_rx_register_callback (hal_spi_sel_en inst, void(*new_callback)(void))
  Actualizar callback en RXRDY.

4.8.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico SPI (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.8.2. Documentación de las estructuras de datos

4.8.2.1. struct hal_spi_master_mode_tx_config_t

Campos de datos

hal_spi_clock_mode_en	clock_mode	
uint16_t	clock_div	

4.8.2.2. 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	
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	

4.8.3. Documentación de las funciones

4.8.3.1. 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

4.8.3.2. 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

4.8.3.3. hal_spi_master_mode_tx_config()

```
void hal_spi_master_mode_tx_config (
            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

4.8.3.4. 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

4.8.3.5. hal_spi_master_mode_tx_register_callback()

```
void hal_spi_master_mode_tx_register_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

4.8.3.6. hal_spi_master_mode_rx_register_callback()

```
void hal_spi_master_mode_rx_register_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

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

Declaraciones a nivel de aplicacion del periferico SYSCON (LPC845)

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

Dependencia gráfica adjunta para HAL_SYSCON.h:

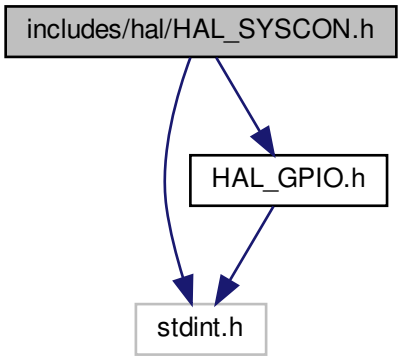
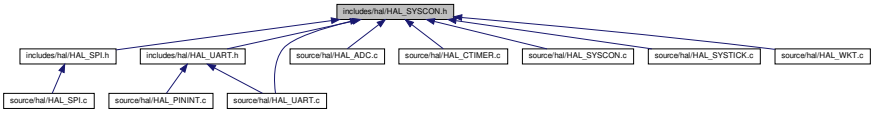


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Enumeraciones

- enum hal_syscon_system_clock_sel_en {
  HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO = 0,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_EXT,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_WATCHDOG,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO_DIV,
  HAL_SYSCON_SYSTEM_CLOCK_SEL_PLL }
- 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_watchdog_clkana_sel_en {
  HAL_SYSCON_WATCHDOG_CLKANA_0KHZ = 0,
  HAL_SYSCON_WATCHDOG_CLKANA_600KHZ,
  HAL_SYSCON_WATCHDOG_CLKANA_1050KHZ,

HAL_SYSCON_WATCHDOG_CLKANA_1400KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_1750KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_2100KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_2400KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_3000KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_3250KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_3500KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_3750KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_4000KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_4200KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_4400KHZ,
HAL_SYSCON_WATCHDOG_CLKANA_4600KHZ }

- 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_system_clock_get (void)

  Obtener la frecuencia actual del main clock.
- void hal_syscon_system_clock_set_source (hal_syscon_system_clock_sel_en clock_source)

  Configuración de fuente de clock para el clock principal.
- void hal_syscon_system_clock_set_divider (uint8_t div)

  Fijar el divisor del clock principal.
- uint32_t hal_syscon_fro_clock_get (void)

  Obtener la frecuencia actual del FRO.

- void hal_syscon_external_crystal_config (uint32_t crystal_freq)

  Configurar el ext clock a partir de un cristal externo.
- void hal_syscon_external_clock_config (uint32_t external_clock_freq)

  Configurar el ext clock a partir de una fuente de clock externa.
- void hal_syscon_fro_clock_config (uint8_t direct)

  Configurar el clock FRO.
- void hal_syscon_fro_clock_disable (void)

  Inhabilitar el FRO.
- void hal_syscon_clkout_config (hal_gpio_portpin_en portpin, 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_frg_config (uint8_t inst, hal_syscon_frg_clock_sel_en clock_source, uint32_t mul)

  Configurar el divisor fraccional.
- void hal_syscon_watchdog_oscillator_config (hal_syscon_watchdog_clkana_sel_en clkana_sel, uint8_t div)

  Configuración del watchdog oscillator.
- uint32_t hal_syscon_peripheral_clock_get (hal_syscon_peripheral_sel_en peripheral)

  Obtener la frecuencia de clock en Hz configurada para cierto periférico.
- void hal_syscon_iocon_glitch_divider_set (hal_syscon_iocon_glitch_sel_en sel, uint32_t div)

  Configurar divisor para el clock de glitches del IOCON.
- void hal_syscon_pll_clock_config (hal_syscon_pll_source_sel_en clock_source, uint32_t freq)

  Configurar el PLL.
- uint32_t hal_syscon_pll_clock_get (void)

  Obtener frecuencia actual configurada del PLL.

4.9.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico SYSCON (LPC845)

Autor

Augusto Santini

Fecha

6/2019

Versión

1.0

4.10. 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:

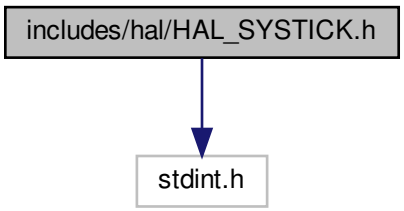
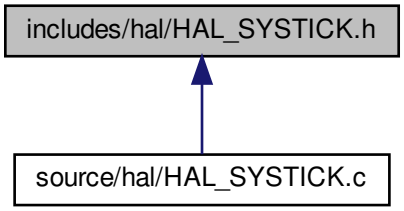


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



typedefs

- typedef void(∗ hal_systick_callback_t) (void)

Funciones

- void hal_systick_init (uint32_t tick_us, void(∗callback)(void))

  Inicializacion del SYSTICK.
- void hal_systick_update_callback (hal_systick_callback_t callback)

  Actualizar callback del SYSTICK.
- void hal_systick_inhibit_set (void)

  Inhabilitar interrupciones de SYSTICK.
- void hal_systick_inhibit_clear (void)

  Habilitar interrupciones de SYSTICK.

4.10.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico SYSICK (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.11. 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:

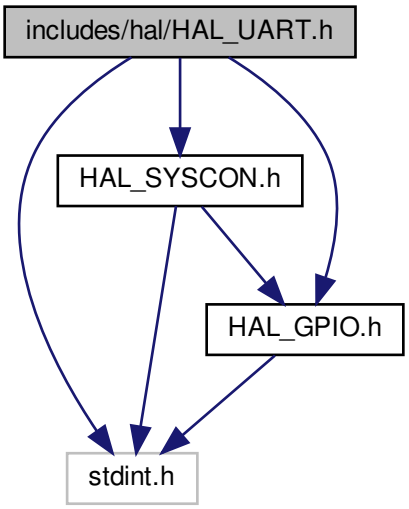
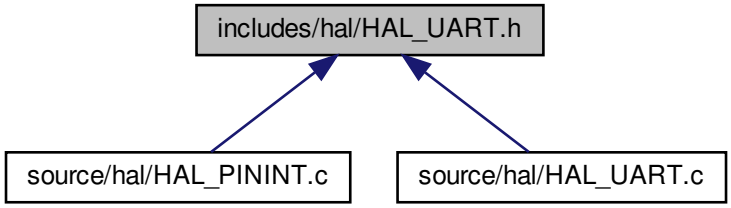


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct hal_uart_config_t

Enumeraciones

- enum **hal_uart_datalen_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_data (uint8_t inst, uint32_t data)

  Transmitir un dato mediante la UART.
- hal_uart_rx_result hal_uart_rx_data (uint8_t inst, uint32_t *data)

  Recibir un dato de la UART.
- void hal_uart_tx_register_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_rx_register_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.

4.11.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico UART (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.11.2. Documentación de las funciones

4.11.2.1. 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

4.11.2.2. hal_uart_tx_data()

```
hal_uart_tx_result hal_uart_tx_data (
            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

4.11.2.3. hal_uart_rx_data()

```
hal_uart_rx_result hal_uart_rx_data (
            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

4.11.2.4. hal_uart_tx_register_callback()

```
void hal_uart_tx_register_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

4.11.2.5. hal_uart_rx_register_callback()

```
void hal_uart_rx_register_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

4.11.2.6. UART3_irq()

```
void UART3_irq (
            void  )
```

Interrupcion de UART3.

4.11.2.7. UART4_irq()

```
void UART4_irq (
            void  )
```

Interrupcion de UART4.

4.12. 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:

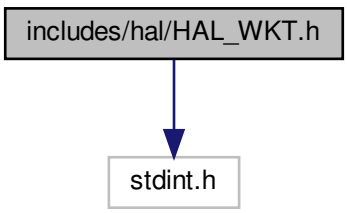
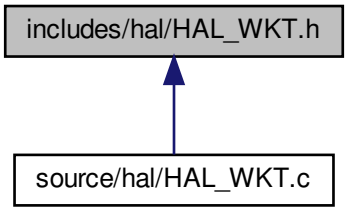


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



typedefs

- typedef void(∗ hal_wkt_callback_t) (void)

  Tipo de dato para el callback de interrupción del WKT.

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, hal_wkt_callback_t callback)

  Inicializar el WKT.
- void hal_wkt_select_clock_source (hal_wkt_clock_source_en clock_sel, uint32_t ext_clock_value)

  Configurar fuente de clock para el WKT.
- void hal_wkt_register_callback (hal_wkt_callback_t new_callback)

  Registrar un callback para la interrupción del WKT.
- void hal_wkt_start_count (uint32_t time_useg)

  Iniciar el conteo con el WKT en base a un tiempo.
- void hal_wkt_start_count_with_value (uint32_t value)

  Iniciar el conteo con el WKT en base a un valor.

4.12.1. Descripción detallada

Declaraciones a nivel de aplicacion del periferico WKT (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

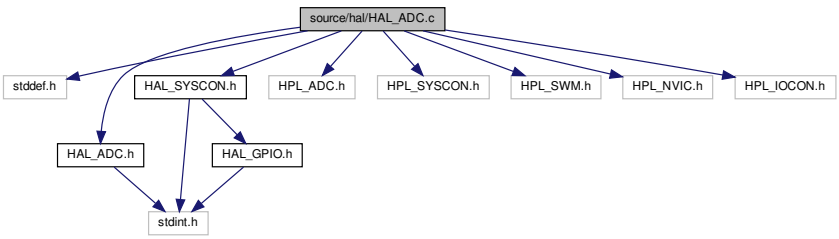
1.0

4.13. Referencia del Archivo source/hal/HAL_ADC.c

Funciones a nivel de aplicacion del periferico ADC (LPC845)

```
#include <stddef.h>
#include <HAL_ADC.h>
#include <HAL_SYSCON.h>
#include <HPL_ADC.h>
#include <HPL_SYSCON.h>
#include <HPL_SWM.h>
#include <HPL_NVIC.h>
#include <HPL_IOCON.h>
```

Dependencia gráfica adjunta para HAL_ADC.c:



Estructuras de datos

- struct flag_sequence_burst_mode_t

defines

- #define ADC_MAX_FREQ_SYNC ((uint32_t) 1.2e6)
- #define ADC_MAX_FREQ_ASYNC ((uint32_t) 0.6e6)
- #define ADC_CYCLE_DELAY (25)
- #define ADC_CHANNEL_AMOUNT (12)

Funciones

- static void dummy_irq_callback (void)

  Funcion dummy para usar como default para las interrupciones.
- 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_sequence_config (hal_adc_sequence_sel_en sequence, const hal_adc_sequence_config_t *config)

  Configurar una secuencia de conversión.
- void hal_adc_sequence_start (hal_adc_sequence_sel_en sequence)

  Disparar conversiones en una secuencia.
- void hal_adc_sequence_stop (hal_adc_sequence_sel_en sequence)

  Detener conversiones en una secuencia de conversión.
- hal_adc_sequence_result_en hal_adc_sequence_get_result (hal_adc_sequence_sel_en sequence, hal_adc_sequence_result_t *result)

  Obtener resultado de la secuencia.
- void hal_adc_threshold_config (hal_adc_threshold_sel_en threshold, uint16_t low, uint16_t high)

  Configurar valor de umbral de comparación.
- void hal_adc_threshold_channel_config (uint8_t adc_channel, hal_adc_threshold_sel_en threshold, hal_adc_threshold_interrupt_sel_en irq_mode)

  Configura un canal para utilizar la funcionalidad de comparación con un umbral y su tipo de interrupción deseada.
- void hal_adc_threshold_register_interrupt (void(*callback)(void))

  Registrar un callabck de interrupción para interrupción por threshold.
- void hal_adc_threshold_get_comparison_results (hal_adc_channel_compare_result_t *results)

  Obtener resultados de comparación de la última conversión.
- void ADC_SEQA_IRQHandler (void)

  Función de interrupción cuando termina la secuencia de conversión A del ADC.
- void ADC_SEQB_IRQHandler (void)

  Función de interrupción cuando termina la secuencia de conversión B del ADC.
- void ADC_THCMP_IRQHandler (void)

  Función de interrupción cuando se detecta alguna de las condiciones de threshold establecidas.
- void ADC_OVR_IRQHandler (void)

  Función de interrupción cuando se detecta alguna de las condiciones de overrun.

Variables

- static void(* adc_seq_completed_callback [2])(void)
- static void(* adc_overrun_callback)(void) = dummy_irq_callback

  Callback cuando ocurre un overrun.
- static void(* adc_compare_callback)(void) = dummy_irq_callback

  Callbacks para las comparaciones de ADC.
- static flag_sequence_burst_mode_t flag_seq_burst_mode

4.13.1. Descripción detallada

Funciones a nivel de aplicacion del periferico ADC (LPC845)

Autor

Augusto Santini
Esteban E. Chiama

Fecha

3/2020

Versión

1.0

4.13.2. Documentación de las estructuras de datos

4.13.2.1. struct flag_sequence_burst_mode_t

Flags para determinar si cada secuencia fue configurada en modo burst o no

Campos de datos

uint8_t	SEQA_burst: 1	Flag burst para secuencia A
uint8_t	SEQB_burst: 1	Flag burst para secuencia B

4.13.3. Documentación de los 'defines'

4.13.3.1. ADC_MAX_FREQ_SYNC

```
#define ADC_MAX_FREQ_SYNC ((uint32_t) 1.2e6)
```

Máxima frecuencia de conversión admitida por el ADC (modo sincrónico)

4.13.3.2. ADC_MAX_FREQ_ASYNC

```
#define ADC_MAX_FREQ_ASYNC ((uint32_t) 0.6e6)
```

Máxima frecuencia de conversión admitida por el ADC (modo asincrónico)

4.13.3.3. ADC_CYCLE_DELAY

```
#define ADC_CYCLE_DELAY (25)
```

Cantidad de ciclos de clock necesarios por el *ADC* para generar una conversión

4.13.3.4. ADC_CHANNEL_AMOUNT

```
#define ADC_CHANNEL_AMOUNT (12)
```

Cantidad de canales disponibles en el *ADC*

4.13.4. Documentación de las funciones

4.13.4.1. dummy_irq_callback()

```
static void dummy_irq_callback (
            void  )  [static]
```

Funcion dummy para usar como default para las interrupciones.

4.13.4.2. ADC_SEQA_IRQHandler()

```
void ADC_SEQA_IRQHandler (
            void  )
```

Función de interrupción cuando termina la secuencia de conversión A del *ADC*.

4.13.4.3. ADC_SEQB_IRQHandler()

```
void ADC_SEQB_IRQHandler (
            void  )
```

Función de interrupción cuando termina la secuencia de conversión B del *ADC*.

4.13.4.4. ADC_THCMP_IRQHandler()

```
void ADC_THCMP_IRQHandler (
            void  )
```

Función de interrupción cuando se detecta alguna de las condiciones de threshold establecidas.

4.13.4.5. ADC_OVR_IRQHandler()

```
void ADC_OVR_IRQHandler (
            void  )
```

Función de interrupción cuando se detecta alguna de las condiciones de overrun.

4.13.5. Documentación de las variables

4.13.5.1. adc_seq_completed_callback

```
void(* adc_seq_completed_callback[2])(void)  [static]
```

Valor inicial:

```
=
{
    dummy_irq_callback,
    dummy_irq_callback
}
```

Callback cuando terminan las secuencias de conversión

4.13.5.2. adc_overrun_callback

```
void(* adc_overrun_callback) (void) = dummy_irq_callback  [static]
```

Callback cuando ocurre un overrun.

4.13.5.3. adc_compare_callback

```
void(* adc_compare_callback) (void) = dummy_irq_callback  [static]
```

Callbacks para las comparaciones de ADC.

4.13.5.4. flag_seq_burst_mode

```
flag_sequence_burst_mode_t flag_seq_burst_mode  [static]
```

Valor inicial:

```
=
{
    .SEQA_burst = 0,
    .SEQB_burst = 0
}
```

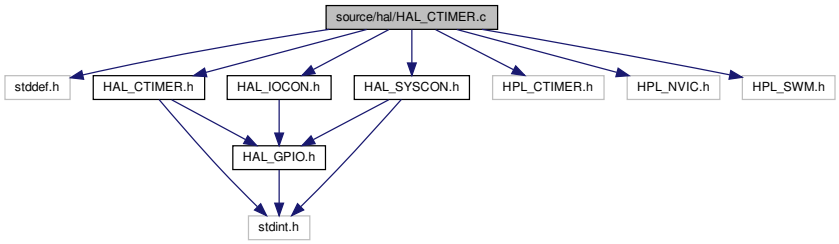
Variable para tener la configuración de modo burst de cada secuencia

4.14. Referencia del Archivo source/hal/HAL_CTIMER.c

Funciones a nivel de aplicacion del periferico CTIMER (LPC845)

```
#include <stddef.h>
#include <HAL_CTIMER.h>
#include <HAL_SYSCON.h>
#include <HAL_IOCON.h>
#include <HPL_CTIMER.h>
#include <HPL_NVIC.h>
#include <HPL_SWM.h>
```

Dependencia gráfica adjunta para HAL_CTIMER.c:



defines

- #define **MATCH_AMOUNT** 4
- #define **CAPTURE_AMOUNT** 4
- #define **PWM_CHANNELS** 3

Funciones

- static void dummy_irq (void)

  Funcion dummy para inicializar los callbacks de interrupcion.
- static uint32_t hal_ctimer_calc_match_value (uint32_t match_value_useg)

  Calcular el valor que debe ir en el registro de match a partir de un valor de useg deseado.
- void hal_ctimer_timer_mode_init (uint32_t clock_div)

  Inicializacion del periferico en modo timer.

- void hal_ctimer_timer_mode_match_config (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_timer_mode_match_change_value (hal_ctimer_match_sel_en match, uint32_t match_value_useg)

  Cambia el valor de MATCH del CTIMER seleccionado.
- uint8_t hal_ctimer_match_read_output (hal_ctimer_match_sel_en match)

  Leer estado de match externo.
- void hal_ctimer_match_set_output (hal_ctimer_match_sel_en match)

  Pone la señal de salida EM# (External Match #) en 1.
- void hal_ctimer_match_clear_output (hal_ctimer_match_sel_en match)

  Pone la señal de salida EM# (External Match #) en 0.
- void hal_ctimer_pwm_mode_init (const hal_ctimer_pwm_config_t *config)

  Inicializar el CTIMER en modo PWM.
- void hal_ctimer_pwm_mode_period_set (uint32_t period_useg)

  Actualizar el periodo en modo PWM.
- void hal_ctimer_pwm_mode_channel_config (hal_ctimer_pwm_channel_sel_en channel_sel, const hal_ctimer_pwm_channel_config_t *channel_config)

  Actualizar configuracion de algun canal de PWM.
- void CTIMER0_IRQHandler (void)

  Interrupcion de CTIMER.

Variables

- void(* match_callbacks [MATCH_AMOUNT])(void)

  Callbacks para interrupciones de match.
- void(* capture_callbacks [CAPTURE_AMOUNT])(void)

  Callbacks para interrupciones de capture.

4.14.1. Descripción detallada

Funciones a nivel de aplicacion del periferico CTIMER (LPC845)

Autor

Augusto Santini

Esteban E. Chiama

Fecha

3/2020

Versión

1.0

4.14.2. Documentación de las funciones

4.14.2.1. dummy_irq()

```
static void dummy_irq (
            void  )  [static]
```

Funcion dummy para inicializar los callbacks de interrupcion.

4.14.2.2. hal_ctimer_calc_match_value()

```
static uint32_t hal_ctimer_calc_match_value (
            uint32_t match_value_useg )  [static]
```

Calcular el valor que debe ir en el registro de match a partir de un valor de useg deseado.

Devuelve

Valor que debe ir en el registro de match

4.14.2.3. 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)

4.14.2.4. hal_ctimer_timer_mode_match_config()

```
void hal_ctimer_timer_mode_match_config (
            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

4.14.2.5. hal_ctimer_timer_mode_run()

```
void hal_ctimer_timer_mode_run (
            void  )
```

Habilitar el conteo del ctimer.

4.14.2.6. hal_ctimer_timer_mode_stop()

```
void hal_ctimer_timer_mode_stop (
            void  )
```

Inhabilitar el conteo del ctimer.

4.14.2.7. hal_ctimer_timer_mode_reset()

```
void hal_ctimer_timer_mode_reset (
            void  )
```

Reiniciar el conteo del ctimer.

4.14.2.8. hal_ctimer_timer_mode_match_change_value()

```
void hal_ctimer_timer_mode_match_change_value (
            hal_ctimer_match_sel_en match,
            uint32_t match_value_useg )
```

Cambia el valor de MATCH del CTIMER seleccionado.

Si el match deseado está configurado para realizar *reload on match*, se escribe el nuevo valor de match será una actualización efectiva en cuanto el conteo actual alcance dicho match. Caso contrario, la actualización del valor de match es inmediata.

Parámetros

in	*match_sel*	Match a configurar
in	*match_value_useg*	Nuevo valor de match, en useg, deseado.

4.14.2.9. hal_ctimer_match_read_output()

```
uint8_t hal_ctimer_match_read_output (
            hal_ctimer_match_sel_en match )
```

Leer estado de match externo.

Parámetros

in	*match*	Numero de match externo a consultar

Devuelve

Estado del match actual

4.14.2.10. hal_ctimer_match_set_output()

```
void hal_ctimer_match_set_output (
            hal_ctimer_match_sel_en match )
```

Pone la señal de salida EM# (External Match #) en 1.

Parámetros

in	*match*	Numero de match externo a configurar

4.14.2.11. hal_ctimer_match_clear_output()

```
void hal_ctimer_match_clear_output (
            hal_ctimer_match_sel_en match )
```

Pone la señal de salida EM# (External Match #) en 0.

Pone la señal de salida EMn (External Match n) en 0.

Parámetros

in	*match*	Numero de match externo a configurar

4.14.2.12. 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

4.14.2.13. hal_ctimer_pwm_mode_period_set()

```
void hal_ctimer_pwm_mode_period_set (
            uint32_t period_useg )
```

Actualizar el periodo en modo PWM.

Parámetros

in	*period_useg*	Nuevo periodo deseado en microsegundos

4.14.2.14. hal_ctimer_pwm_mode_channel_config()

```
void hal_ctimer_pwm_mode_channel_config (
            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

4.14.2.15. CTIMER0_IRQHandler()

```
void CTIMER0_IRQHandler (
            void  )
```

Interrupcion de CTIMER.

4.14.3. Documentación de las variables

4.14.3.1. match_callbacks

```
void(* match_callbacks[MATCH_AMOUNT])(void)
```

Valor inicial:

```
=
{
    dummy_irq,
    dummy_irq,
    dummy_irq,
    dummy_irq
}
```

Callbacks para interrupciones de match.

4.14.3.2. capture_callbacks

```
void(* capture_callbacks[CAPTURE_AMOUNT])(void)
```

Valor inicial:

```
=
{
    dummy_irq,
    dummy_irq,
    dummy_irq,
    dummy_irq
}
```

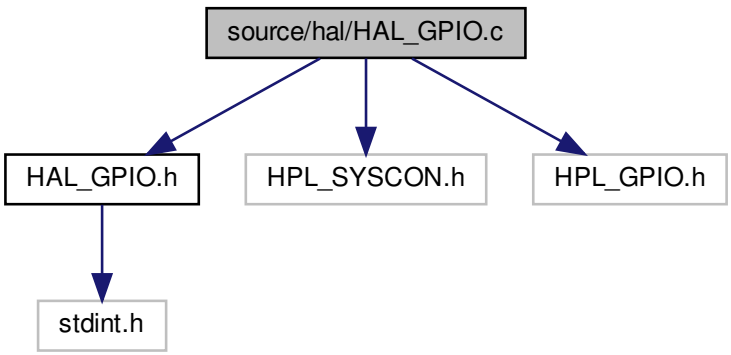
Callbacks para interrupciones de capture.

4.15. Referencia del Archivo source/hal/HAL_GPIO.c

Funciones a nivel de aplicacion del periferico GPIO (LPC845)

```
#include <HAL_GPIO.h>
#include <HPL_SYSCON.h>
#include <HPL_GPIO.h>
```

Dependencia gráfica adjunta para HAL_GPIO.c:



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 dirección de una GPIO.
- void hal_gpio_set_pin (hal_gpio_portpin_en portpin)

  Fijar estado de una GPIO (sin importar máscara)
- void hal_gpio_set_port (hal_gpio_port_en port, uint32_t bits_to_set)

  Fijar estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_set_port (hal_gpio_port_en port, uint32_t bits_to_set)

  Fijar estado de pines de un puerto (teniendo en cuenta máscara)
- void hal_gpio_clear_pin (hal_gpio_portpin_en portpin)

  Limpiar estado de una GPIO (sin importar máscara)
- void hal_gpio_clear_port (hal_gpio_port_en port, uint32_t bits_to_clear)

  Limpiar estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_clear_port (hal_gpio_port_en port, uint32_t bits_to_clear)

  Limpiar estado de pines de un puerto (teniendo en cuenta máscara)
- void hal_gpio_toggle_pin (hal_gpio_portpin_en portpin)

  Invertir estado de una GPIO (sin importar máscara)
- void hal_gpio_toggle_port (hal_gpio_port_en port, uint32_t bits_to_toggle)

  Invertir estado de pines de un puerto (sin importar máscara)
- void hal_gpio_masked_toggle_port (hal_gpio_port_en port, uint32_t bits_to_toggle)

  Invertir estado de pines de un puerto (teniendo en cuenta máscara)
- uint8_t hal_gpio_read_pin (hal_gpio_portpin_en portpin)

  Leer el estado de una GPIO (sin importar máscara)
- uint32_t hal_gpio_read_port (hal_gpio_port_en port)

  Leer estado de un puerto (sin importar máscara)
- uint32_t hal_gpio_masked_read_port (hal_gpio_port_en port)

  Leer estado de un puerto (teniendo en cuenta máscara)
- void hal_gpio_set_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Fijar enmascaramiento de pines en un puerto.
- void hal_gpio_clear_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Limpiar enmascaramiento de pines en un puerto.
- void hal_gpio_toggle_mask_bits (hal_gpio_port_en port, uint32_t mask)

  Invertir enmascaramiento de pines en un puerto.

4.15.1. Descripción detallada

Funciones a nivel de aplicacion del periferico GPIO (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

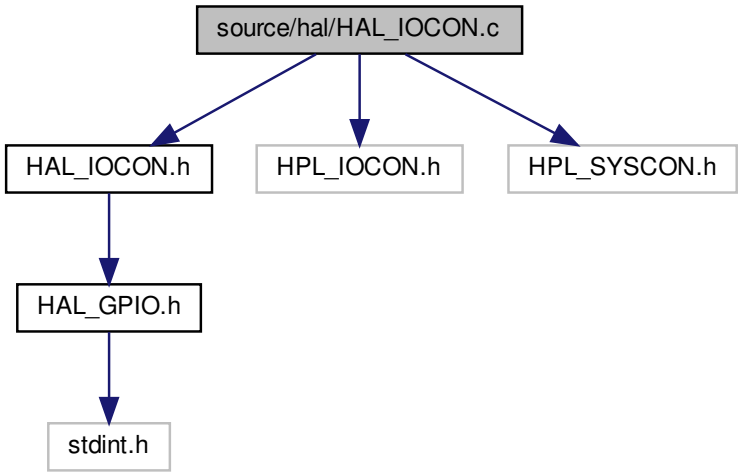
1.0

4.16. Referencia del Archivo source/hal/HAL_IOCON.c

Funciones a nivel de aplicacion del periferico IOCON (LPC845)

```
#include <HAL_IOCON.h>
#include <HPL_IOCON.h>
#include <HPL_SYSCON.h>
```

Dependencia gráfica adjunta para HAL_IOCON.c:



Funciones

- void hal_iocon_config_io (hal_gpio_portpin_en portpin, const hal_iocon_config_t *config)
  Configuracion de un pin.

4.16.1. Descripción detallada

Funciones a nivel de aplicacion del periferico IOCON (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

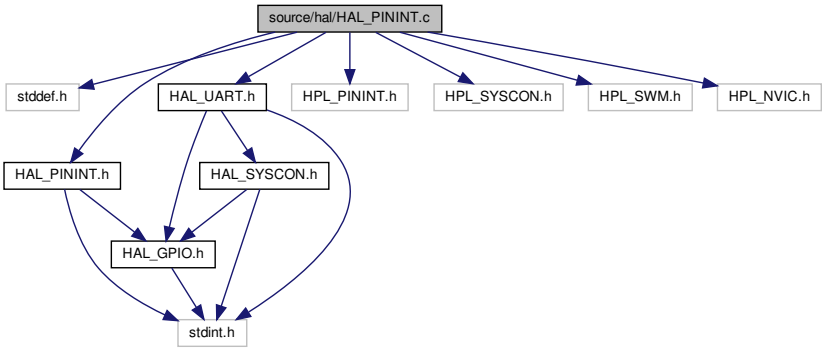
1.0

4.17. Referencia del Archivo source/hal/HAL_PININT.c

Funciones a nivel de aplicacion del periferico PININT (LPC845)

```
#include <stddef.h>
#include <HAL_PININT.h>
#include <HAL_UART.h>
#include <HPL_PININT.h>
#include <HPL_SYSCON.h>
#include <HPL_SWM.h>
#include <HPL_NVIC.h>
```

Dependencia gráfica adjunta para HAL_PININT.c:



defines

- #define PININT_CHANNEL_AMOUNT (8)

Funciones

- static void dummy_irq_callback (void)

  Funcion dummy para inicializar los punteros de interrupciones.
- static void **hal_pinint_enable_channel_irq** (hal_pinint_channel_en channel)
- static void **hal_pinint_disable_channel_irq** (hal_pinint_channel_en channel)
- static void hal_pinint_handle_irq (hal_pinint_channel_en channel)

  Manejo de interrupciones para el modulo.
- void hal_pinint_init (void)

  Inicialización del periférico.
- void hal_pinint_deinit (void)

  De-Inicialización del periférico.
- void hal_pinint_channel_config (hal_pinint_channel_en channel, hal_gpio_portpin_en portpin, hal_pinint_callback_t callback)

  Configuración de canal de PININT.
- void hal_pinint_edge_detections_config (hal_pinint_channel_en channel, hal_pinint_edge_detections_en edge)

  Configurar detecciones por flanco.

- void hal_pinint_level_detections_config (hal_pinint_channel_en channel, hal_pinint_level_detections_en level)

  Configurar detecciones por nivel.
- void PININT0_IRQHandler (void)

  Interrupción para PININT0.
- void PININT1_IRQHandler (void)

  Interrupción para PININT1.
- void PININT2_IRQHandler (void)

  Interrupción para PININT2.
- void PININT3_IRQHandler (void)

  Interrupción para PININT3.
- void PININT4_IRQHandler (void)

  Interrupción para PININT4.
- void PININT5_IRQHandler (void)

  Interrupción para PININT5.
- void PININT6_IRQHandler (void)

  Interrupción para PININT6 y USART3.
- void PININT7_IRQHandler (void)

  Interrupción para PININT7 y USART4.

Variables

- static void(∗ **pinint_callbacks** [PININT_CHANNEL_AMOUNT])(void)

4.17.1. Descripción detallada

Funciones a nivel de aplicacion del periferico PININT (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.17.2. Documentación de los 'defines'

4.17.2.1. PININT_CHANNEL_AMOUNT

```
#define PININT_CHANNEL_AMOUNT (8)
```

Cantidad de canales de *PININT* disponibles

4.17.3. Documentación de las funciones

4.17.3.1. dummy_irq_callback()

```
static void dummy_irq_callback (
            void  )  [static]
```

Funcion dummy para inicializar los punteros de interrupciones.

4.17.3.2. hal_pinint_handle_irq()

```
static void hal_pinint_handle_irq (
            hal_pinint_channel_en channel )  [static]
```

Manejo de interrupciones para el modulo.

Parámetros

in	*channel*	Canal que generó la itnerrupción

4.17.3.3. PININT0_IRQHandler()

```
void PININT0_IRQHandler (
            void  )
```

Interrupción para PININT0.

4.17.3.4. PININT1_IRQHandler()

```
void PININT1_IRQHandler (
            void  )
```

Interrupción para PININT1.

4.17.3.5. PININT2_IRQHandler()

```
void PININT2_IRQHandler (
            void  )
```

Interrupción para PININT2.

4.17.3.6. PININT3_IRQHandler()

```
void PININT3_IRQHandler (
            void  )
```

Interrupción para PININT3.

4.17.3.7. PININT4_IRQHandler()

```
void PININT4_IRQHandler (
            void  )
```

Interrupción para PININT4.

4.17.3.8. PININT5_IRQHandler()

```
void PININT5_IRQHandler (
            void  )
```

Interrupción para PININT5.

4.17.3.9. PININT6_IRQHandler()

```
void PININT6_IRQHandler (
            void  )
```

Interrupción para PININT6 y USART3.

4.17.3.10. PININT7_IRQHandler()

```
void PININT7_IRQHandler (
            void  )
```

Interrupción para PININT7 y USART4.

4.17.4. Documentación de las variables

4.17.4.1. pinint_callbacks

```
void(* pinint_callbacks[PININT_CHANNEL_AMOUNT])(void)  [static]
```

Valor inicial:

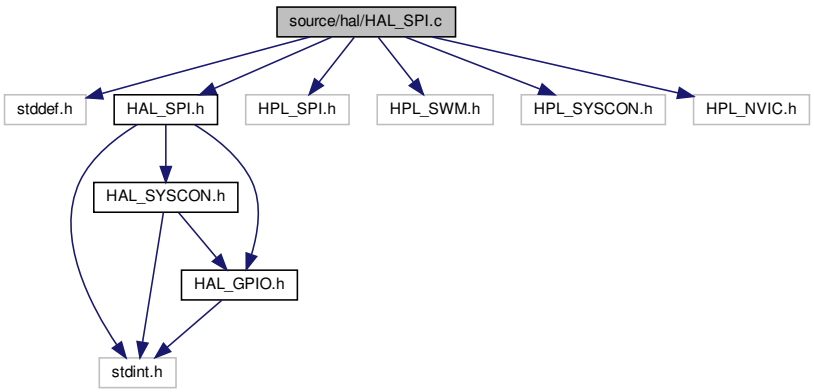
```
= {
        dummy_irq_callback,
        dummy_irq_callback,
        dummy_irq_callback,
        dummy_irq_callback,
        dummy_irq_callback,
        dummy_irq_callback,
        dummy_irq_callback
}
```

4.18. Referencia del Archivo source/hal/HAL_SPI.c

Funciones a nivel de aplicacion del periferico SPI (LPC845)

```
#include <stddef.h>
#include <HAL_SPI.h>
#include <HPL_SPI.h>
#include <HPL_SWM.h>
#include <HPL_SYSCON.h>
#include <HPL_NVIC.h>
```

Dependencia gráfica adjunta para HAL_SPI.c:



Funciones

- static void **dummy_irq** (void)
- static void spi_irq_handler (uint8_t inst)

  Manejador generico de interrupciones de SPI.
- 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_tx_config (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_tx_register_callback (hal_spi_sel_en inst, void(∗new_callback)(void))

  Actualizar callback en TXRDY.
- void hal_spi_master_mode_rx_register_callback (hal_spi_sel_en inst, void(∗new_callback)(void))

  Actualizar callback en RXRDY.
- void SPI0_IRQHandler (void)

  Manejador de interrupcion de SPI0.
- void SPI1_IRQHandler (void)

  Manejador de interrupcion de SPI1.

Variables

- static void(∗ spi_rx_callback [ ])(void)

  Callbacks registrados a la recepcion de un dato por SPI.
- static void(∗ spi_tx_callback [ ])(void)

  Callbacks registrados a la liberacion del buffer de transmision de SPI.

4.18.1. Descripción detallada

Funciones a nivel de aplicacion del periferico SPI (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.18.2. Documentación de las funciones

4.18.2.1. spi_irq_handler()

```
static void spi_irq_handler (
            uint8_t inst ) [static]
```

Manejador generico de interrupciones de SPI.

Parámetros

in	*inst*	Instancia que genero la interrupcion

4.18.2.2. 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

4.18.2.3. 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

4.18.2.4. hal_spi_master_mode_tx_config()

```
void hal_spi_master_mode_tx_config (
            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

4.18.2.5. 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

4.18.2.6. hal_spi_master_mode_tx_register_callback()

```
void hal_spi_master_mode_tx_register_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

4.18.2.7. hal_spi_master_mode_rx_register_callback()

```
void hal_spi_master_mode_rx_register_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

4.18.2.8. SPI0_IRQHandler()

```
void SPI0_IRQHandler (
             void  )
```

Manejador de interrupcion de SPI0.

4.18.2.9. SPI1_IRQHandler()

```
void SPI1_IRQHandler (
             void  )
```

Manejador de interrupcion de SPI1.

4.18.3. Documentación de las variables

4.18.3.1. spi_rx_callback

```
void(* spi_rx_callback[])(void)  [static]
```

Valor inicial:

```
=
{
    dummy_irq,
    dummy_irq
}
```

Callbacks registrados a la recepcion de un dato por SPI.

4.18.3.2. spi_tx_callback

```
void(* spi_tx_callback[])(void)  [static]
```

Valor inicial:

```
=
{
    dummy_irq,
    dummy_irq
}
```

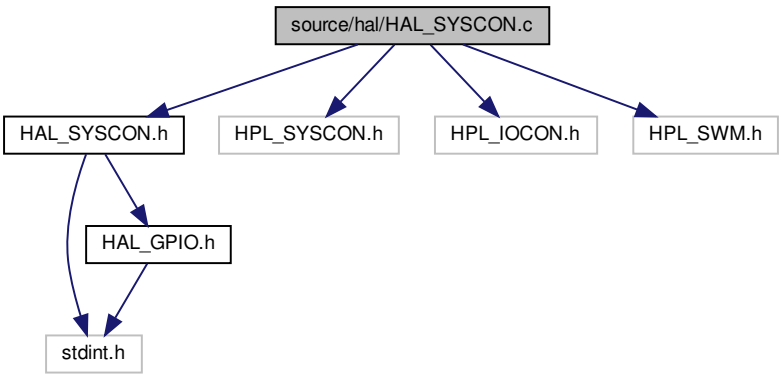
Callbacks registrados a la liberacion del buffer de transmision de SPI.

4.19. Referencia del Archivo source/hal/HAL_SYSCON.c

Funciones a nivel de aplicacion para el SYSCON (LPC845)

```
#include <HAL_SYSCON.h>
#include <HPL_SYSCON.h>
#include <HPL_IOCON.h>
#include <HPL_SWM.h>
```

Dependencia gráfica adjunta para HAL_SYSCON.c:



defines

- #define XTALIN_PORT 0
- #define XTALIN_PIN 8
- #define XTALOUT_PORT 0
- #define XTALOUT_PIN 9
- #define FRO_DIRECT_FREQ 24e6

Funciones

- uint32_t hal_syscon_system_clock_get (void)

  Obtener la frecuencia actual del main clock.
- void hal_syscon_system_clock_set_source (hal_syscon_system_clock_sel_en clock_source)

  Configuración de fuente de clock para el clock principal.
- void hal_syscon_system_clock_set_divider (uint8_t div)

  Fijar el divisor del clock principal.
- uint32_t hal_syscon_fro_clock_get (void)

  Obtener la frecuencia actual del FRO.
- void hal_syscon_external_crystal_config (uint32_t crystal_freq)

  Configurar el ext clock a partir de un cristal externo.
- void hal_syscon_external_clock_config (uint32_t external_clock_freq)

  Configurar el ext clock a partir de una fuente de clock externa.
- void hal_syscon_fro_clock_config (uint8_t direct)

  Configurar el clock FRO.

- void hal_syscon_fro_clock_disable (void)
  Inhabilitar el FRO.
- void hal_syscon_clkout_config (hal_gpio_portpin_en portpin, 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_frg_config (uint8_t inst, hal_syscon_frg_clock_sel_en clock_source, uint32_t mul)
  Configurar el divisor fraccional.
- void hal_syscon_watchdog_oscillator_config (hal_syscon_watchdog_clkana_sel_en clkana_sel, uint8_t div)
  Configuración del watchdog oscillator.
- uint32_t hal_syscon_peripheral_clock_get (hal_syscon_peripheral_sel_en peripheral)
  Obtener la frecuencia de clock en Hz configurada para cierto periférico.
- void hal_syscon_iocon_glitch_divider_set (hal_syscon_iocon_glitch_sel_en sel, uint32_t div)
  Configurar divisor para el clock de glitches del IOCON.
- void hal_syscon_pll_clock_config (hal_syscon_pll_source_sel_en clock_source, uint32_t freq)
  Configurar el PLL.
- uint32_t hal_syscon_pll_clock_get (void)
  Obtener frecuencia actual configurada del PLL.

Variables

- static uint8_t current_main_div = 1
  Divisor actual del clock principal.
- static uint32_t current_fro_freq = FRO_DIRECT_FREQ / 2
  Frecuencia actual del FRO.
- static uint32_t current_fro_div_freq = FRO_DIRECT_FREQ / 4
  Frecuencia actual del FRO DIV.
- static uint32_t current_crystal_freq = 0
  Frecuencia del cristal configurada.
- static uint32_t current_frg_freq [2] = { 0, 0 }
  Frecuencia de los FRG.
- static uint32_t current_pll_freq = 0
  Frecuencia del PLL.
- static uint32_t current_ext_freq = 0
  Frecuencia de la fuente de clock externa.
- static uint32_t current_watchdog_freq = 0
  Frecuencia del watchod oscillator.
- static uint32_t * current_main_freq = &current_fro_freq
  Frecuencia actual del main clock.
- static const uint32_t base_watchdog_freq [ ]
  Frecuencias bases posibles del watchod oscillator.

4.19.1. Descripción detallada

Funciones a nivel de aplicacion para el SYSCON (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.19.2. Documentación de los 'defines'

4.19.2.1. XTALIN_PORT

```
#define XTALIN_PORT 0
```

Número de puerto del XTALIN

4.19.2.2. XTALIN_PIN

```
#define XTALIN_PIN 8
```

Número de pin del XTALIN

4.19.2.3. XTALOUT_PORT

```
#define XTALOUT_PORT 0
```

Número de puerto del XTALOUT

4.19.2.4. XTALOUT_PIN

```
#define XTALOUT_PIN 9
```

Número de pin del XTALOUT

4.19.2.5. FRO_DIRECT_FREQ

```
#define FRO_DIRECT_FREQ 24e6
```

Frecuencia del FRO base

4.19.3. Documentación de las variables

4.19.3.1. current_main_div

```
uint8_t current_main_div = 1  [static]
```

Divisor actual del clock principal.

4.19.3.2. current_fro_freq

```
uint32_t current_fro_freq = FRO_DIRECT_FREQ / 2  [static]
```

Frecuencia actual del FRO.

4.19.3.3. current_fro_div_freq

```
uint32_t current_fro_div_freq = FRO_DIRECT_FREQ / 4  [static]
```

Frecuencia actual del FRO DIV.

4.19.3.4. current_crystal_freq

```
uint32_t current_crystal_freq = 0  [static]
```

Frecuencia del cristal configurada.

4.19.3.5. current_frg_freq

```
uint32_t current_frg_freq[2] = { 0, 0 }  [static]
```

Frecuencia de los FRG.

4.19.3.6. current_pll_freq

```
uint32_t current_pll_freq = 0  [static]
```

Frecuencia del PLL.

4.19.3.7. current_ext_freq

```
uint32_t current_ext_freq = 0  [static]
```

Frecuencia de la fuente de clock externa.

4.19.3.8. current_watchdog_freq

```
uint32_t current_watchdog_freq = 0  [static]
```

Frecuencia del watchod oscillator.

4.19.3.9. current_main_freq

```
uint32_t* current_main_freq = &current_fro_freq  [static]
```

Frecuencia actual del main clock.

4.19.3.10. base_watchdog_freq

```
const uint32_t base_watchdog_freq[]  [static]
```

Valor inicial:

```
=
{
    0, 0.6e6, 1.05e6, 1.4e6, 1.75e6, 2.1e6, 2.4e6, 2.7e6,
    3e6, 3.25e6, 3.5e6, 3.75e6, 4e6, 4.2e6, 4.4e6, 4.6e6
}
```

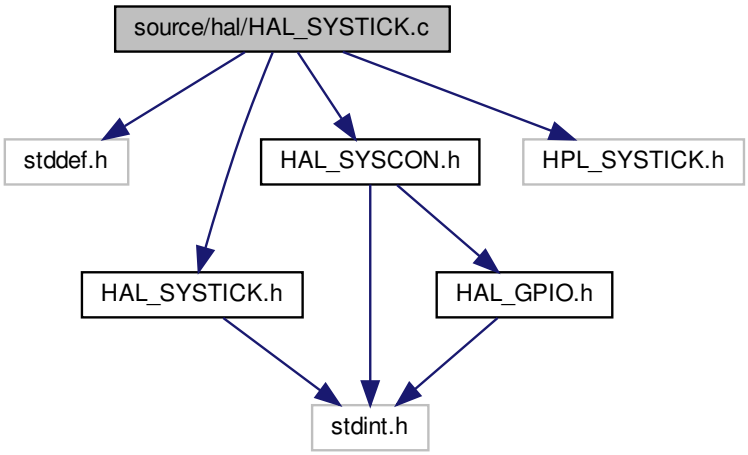
Frecuencias bases posibles del watchod oscillator.

4.20. Referencia del Archivo source/hal/HAL_SYSTICK.c

Funciones a nivel de aplicacion para el SYSTICK (LPC845)

```
#include <stddef.h>
#include <HAL_SYSTICK.h>
#include <HAL_SYSCON.h>
#include <HPL_SYSTICK.h>
```

Dependencia gráfica adjunta para HAL_SYSTICK.c:



Funciones

- static void dummy_irq (void)

  Dummy function para inicializar los punteros a los callbacks.
- 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.
- void hal_systick_inhibit_set (void)

  Inhabilitar interrupciones de SYSTICK.
- void hal_systick_inhibit_clear (void)

  Habilitar interrupciones de SYSTICK.
- void SysTick_Handler (void)

  Interrupcion de SYSTICK.

Variables

- static void(∗ systick_callback)(void) = dummy_irq

  Callback a llamar en la interrupcion.

4.20.1. Descripción detallada

Funciones a nivel de aplicacion para el SYSTICK (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.20.2. Documentación de las funciones

4.20.2.1. dummy_irq()

```
static void dummy_irq (
            void  )  [static]
```

Dummy function para inicializar los punteros a los callbacks.

4.20.2.2. SysTick_Handler()

```
void SysTick_Handler (
            void  )
```

Interrupcion de SYSTICK.

4.20.3. Documentación de las variables

4.20.3.1. systick_callback

```
void(* systick_callback) (void) = dummy_irq  [static]
```

Callback a llamar en la interrupcion.

Ejemplos

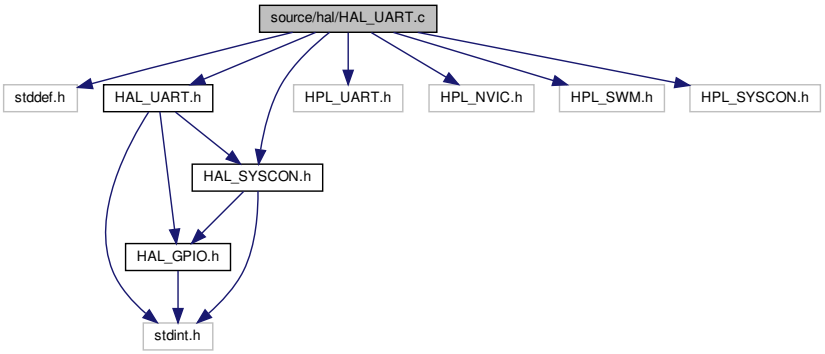
Ejemplo_ADC.c.

4.21. Referencia del Archivo source/hal/HAL_UART.c

Funciones a nivel de aplicacion del periferico UART (LPC845)

```
#include <stddef.h>
#include <HAL_UART.h>
#include <HAL_SYSCON.h>
#include <HPL_UART.h>
#include <HPL_NVIC.h>
#include <HPL_SWM.h>
#include <HPL_SYSCON.h>
```

Dependencia gráfica adjunta para HAL_UART.c:



Funciones

- static void dummy_callback (void)

  Llamado a funcion dummy para irq iniciales.
- static uint16_t hal_uart_calculate_brgval (uint32_t uart_clock, uint32_t baudrate, uint8_t oversampling)

  Calculo del valor para el registro de Baudrate.
- static void **hal_uart_handle_irq** (uint8_t inst)
- 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_data (uint8_t inst, uint32_t data)

  Transmitir un dato mediante la UART.
- hal_uart_rx_result hal_uart_rx_data (uint8_t inst, uint32_t ∗data)

  Recibir un dato de la UART.
- void hal_uart_rx_register_callback (uint8_t inst, void(∗new_callback)(void))

  Registrar el callback a ser llamado en la recepcion de un dato por UART.
- void hal_uart_tx_register_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 UART0_IRQHandler (void)

  Interrupcion de UART0.
- void UART1_IRQHandler (void)

  Interrupcion de UART1.
- void UART2_IRQHandler (void)

  Interrupcion de UART2.
- void UART3_irq (void)

  Interrupcion de UART3.
- void UART4_irq (void)

  Interrupcion de UART4.

Variables

- static void(∗ uart_rx_callback [ ])(void)

  Callbacks registrados a la recepcion de un dato por UART.
- static void(∗ uart_tx_callback [ ])(void)

  Callbacks registrados a la finalizacion de transmision de un dato por UART.

4.21.1. Descripción detallada

Funciones a nivel de aplicacion del periferico UART (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.21.2. Documentación de las funciones

4.21.2.1. dummy_callback()

```
static void dummy_callback (
            void  )  [static]
```

Llamado a funcion dummy para irq iniciales.

4.21.2.2. hal_uart_calculate_brgval()

```
static uint16_t hal_uart_calculate_brgval (
            uint32_t uart_clock,
            uint32_t baudrate,
            uint8_t oversampling )  [inline], [static]
```

Calculo del valor para el registro de Baudrate.

Parámetros

in	*uart_clock*	Clock asociado con la UART.
in	*baudrate*	Baudrate deseado a calcular.
in	*oversampling*	Oversampling deseado para la UART.

Devuelve

Valor a poner en el registro BRG.

4.21.2.3. 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

4.21.2.4. hal_uart_tx_data()

```
hal_uart_tx_result hal_uart_tx_data (
            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

4.21.2.5. hal_uart_rx_data()

```
hal_uart_rx_result hal_uart_rx_data (
            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

4.21.2.6. hal_uart_rx_register_callback()

```
void hal_uart_rx_register_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

4.21.2.7. hal_uart_tx_register_callback()

```
void hal_uart_tx_register_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

4.21.2.8. UART0_IRQHandler()

```
void UART0_IRQHandler (
            void  )
```

Interrupcion de UART0.

4.21.2.9. UART1_IRQHandler()

```
void UART1_IRQHandler (
            void  )
```

Interrupcion de UART1.

4.21.2.10. UART2_IRQHandler()

```
void UART2_IRQHandler (
            void  )
```

Interrupcion de UART2.

4.21.2.11. UART3_irq()

```
void UART3_irq (
            void  )
```

Interrupcion de UART3.

4.21.2.12. UART4_irq()

```
void UART4_irq (
              void  )
```

Interrupcion de UART4.

4.21.3. Documentación de las variables

4.21.3.1. uart_rx_callback

```
void(* uart_rx_callback[])(void)  [static]
```

Valor inicial:

```
=
{
        dummy_callback,
        dummy_callback,
        dummy_callback,
        dummy_callback,
        dummy_callback
}
```

Callbacks registrados a la recepcion de un dato por UART.

4.21.3.2. uart_tx_callback

```
void(* uart_tx_callback[])(void)  [static]
```

Valor inicial:

```
=
{
        dummy_callback,
        dummy_callback,
        dummy_callback,
        dummy_callback,
        dummy_callback
}
```

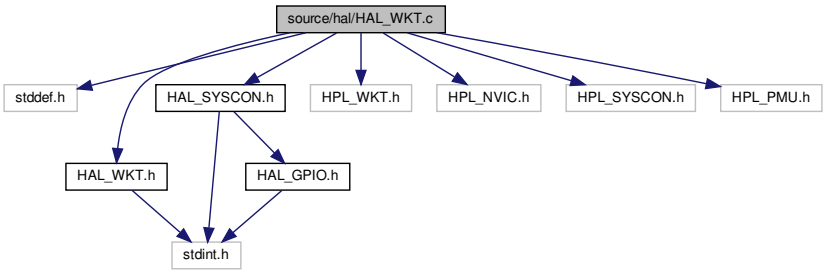
Callbacks registrados a la finalizacion de transmision de un dato por UART.

4.22. Referencia del Archivo source/hal/HAL_WKT.c

Funciones a nivel de aplicacion del periferico WKT (LPC845)

```
#include <stddef.h>
#include <HAL_WKT.h>
#include <HAL_SYSCON.h>
#include <HPL_WKT.h>
#include <HPL_NVIC.h>
#include <HPL_SYSCON.h>
#include <HPL_PMU.h>
```

Dependencia gráfica adjunta para HAL_WKT.c:



defines

- #define HAL_WKT_DIVIDE_VALUE (16)
- #define HAL_WKT_LOW_POWER_OSC_FREQ (10e3)

Funciones

- static void dummy_irq (void)

  Funcion dummy para inicializar punteros a funcion de interrupcion.
- 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)

  Configurar fuente de clock para el WKT.
- void hal_wkt_register_callback (void(∗new_callback)(void))

  Registrar un callback para la interrupción del WKT.
- void hal_wkt_start_count (uint32_t time_useg)

  Iniciar el conteo con el WKT en base a un tiempo.
- void hal_wkt_start_count_with_value (uint32_t value)

  Iniciar el conteo con el WKT en base a un valor.
- void WKT_IRQHandler (void)

  Interrupcion de WKT.

Variables

- static hal_wkt_clock_source_en current_clock_source = HAL_WKT_CLOCK_SOURCE_FRO_DIV
- static uint32_t current_ext_clock = 0
- static hal_wkt_callback_t hal_wkt_irq_callback = dummy_irq

4.22.1. Descripción detallada

Funciones a nivel de aplicacion del periferico WKT (LPC845)

Autor

Augusto Santini

Fecha

3/2020

Versión

1.0

4.22.2. Documentación de los 'defines'

4.22.2.1. HAL_WKT_DIVIDE_VALUE

```
#define HAL_WKT_DIVIDE_VALUE (16)
```

Valor de división que genera el *WKT* al utilizar el *FRO* como fuente de clock

4.22.2.2. HAL_WKT_LOW_POWER_OSC_FREQ

```
#define HAL_WKT_LOW_POWER_OSC_FREQ (10e3)
```

Frecuencia del oscilador de bajo consumo

4.22.3. Documentación de las funciones

4.22.3.1. dummy_irq()

```
static void dummy_irq (
            void  )  [static]
```

Funcion dummy para inicializar punteros a funcion de interrupcion.

4.22.3.2. WKT_IRQHandler()

```
void WKT_IRQHandler (
            void  )
```

Interrupcion de WKT.

4.22.4. Documentación de las variables

4.22.4.1. current_clock_source

```
hal_wkt_clock_source_en current_clock_source = HAL_WKT_CLOCK_SOURCE_FRO_DIV  [static]
```

Fuente actual configurada para el *WKT*

4.22.4.2. current_ext_clock

```
uint32_t current_ext_clock = 0  [static]
```

Frecuencia actual externa configurada para el *WKT*

4.22.4.3. hal_wkt_irq_callback

```
hal_wkt_callback_t hal_wkt_irq_callback = dummy_irq  [static]
```

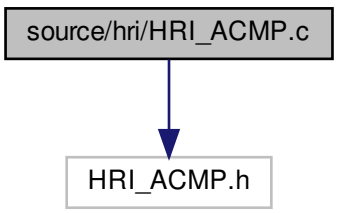
Puntero al callback a ejecutar en interrupción de *WKT*

4.23. Referencia del Archivo source/hri/HRI_ACMP.c

Funciones a nivel de abstraccion de periferico para el ADC (LPC845)

```
#include <HRI_ACMP.h>
```

Dependencia gráfica adjunta para HRI_ACMP.c:



Variables

- volatile ACMP_per_t ∗const ACMP = (ACMP_per_t ∗) ACMP_BASE
  Periferico ANALOG COMPARATOR.

4.23.1. Descripción detallada

Funciones a nivel de abstraccion de periferico para el ADC (LPC845)

Autor

Esteban E. Chiama

Fecha

4/2020

Versión

1.0

4.23.2. Documentación de las variables

4.23.2.1. ACMP

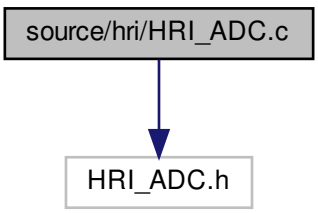
```
volatile ACMP_per_t* const ACMP = (ACMP_per_t *) ACMP_BASE
```

Periferico ANALOG COMPARATOR.

4.24. Referencia del Archivo source/hri/HRI_ADC.c

Declaración del periférico ADC (LPC845)

```
#include <HRI_ADC.h>
```
Dependencia gráfica adjunta para HRI_ADC.c:



Variables

- volatile ADC_per_t ∗const ADC = (ADC_per_t ∗) ADC_BASE
  Periferico ADC.

4.24.1. Descripción detallada

Declaración del periférico ADC (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.24.2. Documentación de las variables

4.24.2.1. ADC

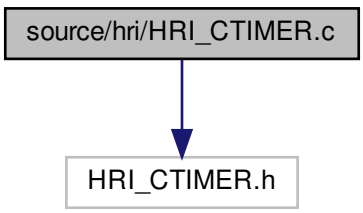
```
volatile ADC_per_t* const ADC = (ADC_per_t *) ADC_BASE
```

Periferico ADC.

4.25. Referencia del Archivo source/hri/HRI_CTIMER.c

Declaración del periférico CTIMER (LPC845)

```
#include <HRI_CTIMER.h>
```
Dependencia gráfica adjunta para HRI_CTIMER.c:



Variables

- volatile CTIMER_per_t ∗const CTIMER = (volatile CTIMER_per_t ∗) CTIMER_BASE
  Periferico CTIMER.

4.25.1. Descripción detallada

Declaración del periférico CTIMER (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.25.2. Documentación de las variables

4.25.2.1. CTIMER

```
volatile CTIMER_per_t* const CTIMER = (volatile CTIMER_per_t *) CTIMER_BASE
```

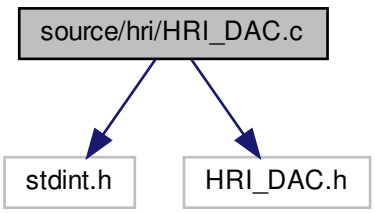
Periferico CTIMER.

4.26. Referencia del Archivo source/hri/HRI_DAC.c

Declaración del periférico DAC (LPC845)

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

Dependencia gráfica adjunta para HRI_DAC.c:



Variables

- volatile DAC_per_t ∗const **DAC** [ ]

4.26.1. Descripción detallada

Declaración del periférico DAC (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.26.2. Documentación de las variables

4.26.2.1. DAC

```
volatile DAC_per_t* const DAC[]
```

Valor inicial:

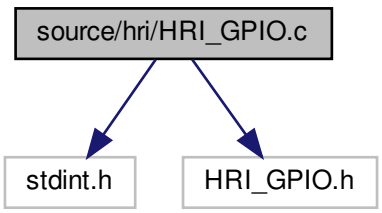
```
= {
        (DAC_per_t *) DAC0_BASE,
        (DAC_per_t *) DAC1_BASE
}
```

4.27. Referencia del Archivo source/hri/HRI_GPIO.c

Declaración del periférico GPIO (LPC845)

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

Dependencia gráfica adjunta para HRI_GPIO.c:



Variables

- volatile GPIO_per_t ∗const GPIO = (GPIO_per_t ∗) GPIO_BASE
  Periferico GPIO.

4.27.1. Descripción detallada

Declaración del periférico GPIO (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.27.2. Documentación de las variables

4.27.2.1. GPIO

```
volatile GPIO_per_t* const GPIO = (GPIO_per_t *) GPIO_BASE
```

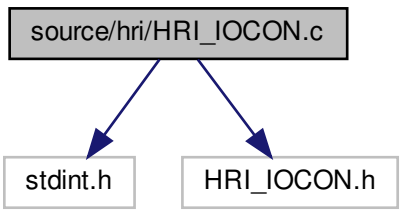
Periferico GPIO.

4.28. Referencia del Archivo source/hri/HRI_IOCON.c

Declaración del periférico IOCON (LPC845)

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

Dependencia gráfica adjunta para HRI_IOCON.c:



Variables

- volatile IOCON_per_t ∗const IOCON = (IOCON_per_t ∗) IOCON_BASE

  Periferico IOCON.
- volatile IOCON_PIO_reg_t dummy_reg

  Registro dummy para los pines no disponibles en el encapsulado.
- volatile IOCON_PIO_reg_t ∗const IOCON_PIN_TABLE [2][32]

  Tabla de registros de configuracion.

4.28.1. Descripción detallada

Declaración del periférico IOCON (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.28.2. Documentación de las variables

4.28.2.1. IOCON

```
volatile IOCON_per_t* const IOCON = (IOCON_per_t *) IOCON_BASE
```

Periferico IOCON.

4.28.2.2. dummy_reg

```
volatile IOCON_PIO_reg_t dummy_reg
```

Registro dummy para los pines no disponibles en el encapsulado.

4.28.2.3. IOCON_PIN_TABLE

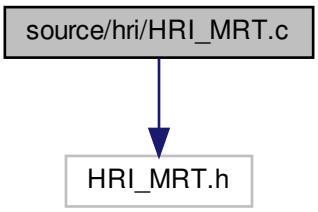
```
volatile IOCON_PIO_reg_t* const IOCON_PIN_TABLE[2][32]
```

Tabla de registros de configuracion.

4.29. Referencia del Archivo source/hri/HRI_MRT.c

Declaración del periférico MRT (LPC845)

```
#include <HRI_MRT.h>
```
Dependencia gráfica adjunta para HRI_MRT.c:



Variables

- volatile MRT_per_t ∗const MRT = (MRT_per_t ∗) MRT_BASE
  Periferico MRT.

4.29.1. Descripción detallada

Declaración del periférico MRT (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.29.2. Documentación de las variables

4.29.2.1. MRT

```
volatile MRT_per_t* const MRT = (MRT_per_t *) MRT_BASE
```

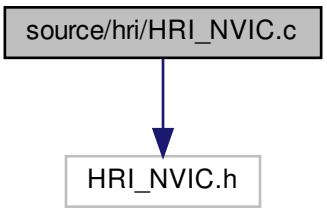
Periferico MRT.

4.30. Referencia del Archivo source/hri/HRI_NVIC.c

Declaración del periférico NVIC (LPC845)

```
#include <HRI_NVIC.h>
```

Dependencia gráfica adjunta para HRI_NVIC.c:



Variables

- volatile NVIC_per_t *const NVIC = (NVIC_per_t *) NVIC_BASE
  Periferico NVIC.

4.30.1. Descripción detallada

Declaración del periférico NVIC (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.30.2. Documentación de las variables

4.30.2.1. NVIC

```
volatile NVIC_per_t* const NVIC = (NVIC_per_t *) NVIC_BASE
```

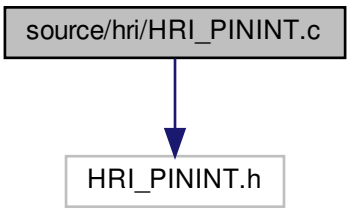
Periferico NVIC.

4.31. Referencia del Archivo source/hri/HRI_PININT.c

Declaración del periférico PININT (LPC845)

```
#include <HRI_PININT.h>
```

Dependencia gráfica adjunta para HRI_PININT.c:



Variables

- volatile PININT_per_t *const PININT = (PININT_per_t *) PININT_BASE
  Periferico PININT.

4.31.1. Descripción detallada

Declaración del periférico PININT (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.31.2. Documentación de las variables

4.31.2.1. PININT

```
volatile PININT_per_t* const PININT = (PININT_per_t *) PININT_BASE
```

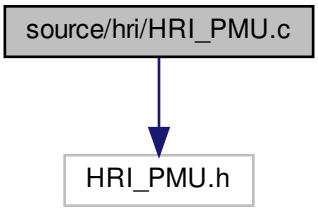
Periferico PININT.

4.32. Referencia del Archivo source/hri/HRI_PMU.c

Declaración del periférico PMU (LPC845)

```
#include <HRI_PMU.h>
```

Dependencia gráfica adjunta para HRI_PMU.c:



Variables

- volatile SCR_reg_t *const SCR = (volatile SCR_reg_t *) SCR_REG_BASE
  Registro SCR.
- volatile PMU_per_t *const PMU = (volatile PMU_per_t *) PMU_BASE
  Periferico PMU.

4.32.1. Descripción detallada

Declaración del periférico PMU (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.32.2. Documentación de las variables

4.32.2.1. SCR

```
volatile SCR_reg_t* const SCR = (volatile SCR_reg_t *) SCR_REG_BASE
```

Registro SCR.

4.32.2.2. PMU

```
volatile PMU_per_t* const PMU = (volatile PMU_per_t *) PMU_BASE
```

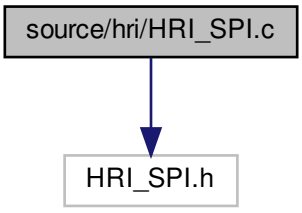
Periferico PMU.

4.33. Referencia del Archivo source/hri/HRI_SPI.c

Declaración del periférico SPI (LPC845)

```
#include <HRI_SPI.h>
```

Dependencia gráfica adjunta para HRI_SPI.c:



Variables

- volatile SPI_per_t ∗const **SPI** [ ]

4.33.1. Descripción detallada

Declaración del periférico SPI (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.33.2. Documentación de las variables

4.33.2.1. SPI

```
volatile SPI_per_t* const SPI[]
```

Valor inicial:

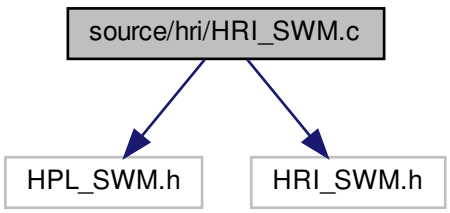
```
= {
        (SPI_per_t *) SPI0_BASE,
        (SPI_per_t *) SPI1_BASE
}
```

4.34. Referencia del Archivo source/hri/HRI_SWM.c

Declaración del periférico SWM (LPC845)

```
#include <HPL_SWM.h>
#include <HRI_SWM.h>
```

Dependencia gráfica adjunta para HRI_SWM.c:



Variables

- volatile SWM_per_t ∗const SWM = (SWM_per_t ∗) SWM_BASE
  Periferico SWM.

4.34.1. Descripción detallada

Declaración del periférico SWM (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.34.2. Documentación de las variables

4.34.2.1. SWM

```
volatile SWM_per_t* const SWM = (SWM_per_t *) SWM_BASE
```

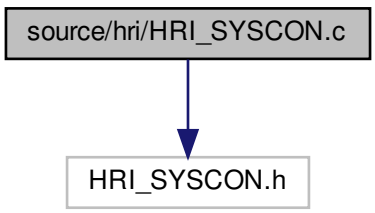
Periferico SWM.

4.35. Referencia del Archivo source/hri/HRI_SYSCON.c

Declaración del periférico SYSCON (LPC845)

```
#include <HRI_SYSCON.h>
```

Dependencia gráfica adjunta para HRI_SYSCON.c:



Variables

- volatile SYSCON_per_t ∗const SYSCON = (SYSCON_per_t ∗) SYSCON_BASE
  Periferico SYSCON.

4.35.1. Descripción detallada

Declaración del periférico SYSCON (LPC845)

Autor

Augusto Santini

Fecha

4/20120

Versión

1.0

4.35.2. Documentación de las variables

4.35.2.1. SYSCON

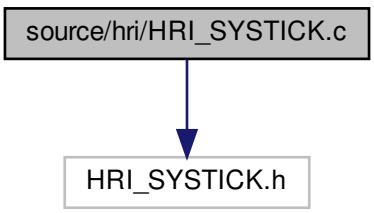
```
volatile SYSCON_per_t* const SYSCON = (SYSCON_per_t *) SYSCON_BASE
```

Periferico SYSCON.

4.36. Referencia del Archivo source/hri/HRI_SYSTICK.c

Declaración del periférico SYSTICK (LPC845)

```
#include <HRI_SYSTICK.h>
```
Dependencia gráfica adjunta para HRI_SYSTICK.c:



Variables

- volatile SYSTICK_reg_t ∗const SYSTICK = (SYSTICK_reg_t ∗) SYSTICK_BASE
  Periferico SYSTICK.

4.36.1. Descripción detallada

Declaración del periférico SYSTICK (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.36.2. Documentación de las variables

4.36.2.1. SYSTICK

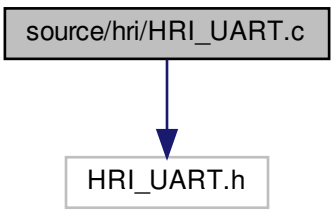
```
volatile SYSTICK_reg_t* const SYSTICK = (SYSTICK_reg_t *) SYSTICK_BASE
```

Periferico SYSTICK.

4.37. Referencia del Archivo source/hri/HRI_UART.c

Declaración del periférico UART (LPC845)

```
#include <HRI_UART.h>
```
Dependencia gráfica adjunta para HRI_UART.c:



Variables

- volatile UART_per_t ∗const **UART** [ ]

4.37.1. Descripción detallada

Declaración del periférico UART (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.37.2. Documentación de las variables

4.37.2.1. UART

```
volatile UART_per_t* const UART[]
```

Valor inicial:

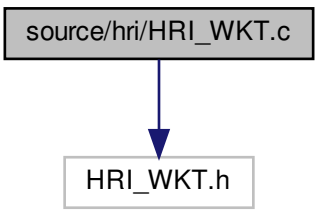
```
= {
        (UART_per_t *) UART0_BASE,
        (UART_per_t *) UART1_BASE,
        (UART_per_t *) UART2_BASE,
        (UART_per_t *) UART3_BASE,
        (UART_per_t *) UART4_BASE
}
```

4.38. Referencia del Archivo source/hri/HRI_WKT.c

Declaración del periférico WKT (LPC845)

```
#include <HRI_WKT.h>
```

Dependencia gráfica adjunta para HRI_WKT.c:



Variables

- volatile WKT_per_t ∗const WKT = (volatile WKT_per_t ∗) WKT_BASE
  Periferico WKT.

4.38.1. Descripción detallada

Declaración del periférico WKT (LPC845)

Autor

Augusto Santini

Fecha

4/2020

Versión

1.0

4.38.2. Documentación de las variables

4.38.2.1. WKT

```
volatile WKT_per_t* const WKT = (volatile WKT_per_t *) WKT_BASE
```

Periferico WKT.

Capítulo 5

Documentación de ejemplos

5.1. Ejemplo_ADC.c

Configuraciones

El programa utiliza el clock por default con el que comienza el microcontrolador, es decir, el *Free Running Oscillator* funcionando a 12MHz

El periférico *ADC* 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 el canal 0:

- El canal 0 está conectado al preset propio del stick de desarrollo (Puerto 0 pin 7)

La secuencia B es configurada para generar conversiones en los canales 4 y 8:

- El canal 4 está ubicado en el pin número 7 (Puerto 0 pin 22)
- El canal 8 está ubicado en el pin número 3 (Puerto 0 pin 18)

En ambos canales de la secuencia B se puede conectar un preset con los terminales de los extremos uno a VDD y el otro a VSS, y el terminal central a cada uno de los canales.

Además, la secuencia A tendrá la siguiente configuración:

- Trigger: Únicamente se disparan conversiones por software
- Bypass sincronismo: Sí
- Modo de interrupción: Cuando termina cada canal
- Burst: Habilitado

- Un trigger dispara: Una conversión de canal
- Secuencia A como baja prioridad: Si

La secuencia B tendrá la siguiente configuración:

- Trigger: Únicamente se disparan conversiones por software
- Bypass sincronismo: Sí
- Modo de interrupción: Cuando se termina la secuencia completa
- Burst: Inhabilitado
- Un trigger dispara: Una conversión de secuencia completa

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

Descripción de funcionamiento

La idea de este ejemplo es demostrar las capacidades de la librería para con el periférico *ADC*. Con dicho fin, se muestran las dos principales capacidades del periférico, incluidas particularidades explicadas a continuación.

Conversiones no continuas

La *secuencia B* está configurada para generar conversiones disparadas por software. Esta forma es usualmente utilizada cuando no es necesario tener conversiones continuas, dada la naturaleza de necesidad de la aplicación a desarrollar. Sin embargo, no es la óptima para distintos casos como pueden ser, por ejemplo:

- Conversiones con tiempos precisos. En estos casos, se recomienda disparar conversiones disparadas por algún timer, la librería tiene implementada esta configuración.
- Conversiones que dependen de variables externas. En estos casos, es útil disparar las conversiones mediante interrupciones de *PININT* o *ACOMP*.

En el caso del ejemplo, se disparan conversiones cada aproximadamente 100 milisegundos con ayuda del *Systick*.

Conversiones continuas

La *secuencia A*, al tener el modo *BURST* habilitado, genera conversiones continuamente. Dependiendo de la necesidad de procesamiento de dichos resultados, tal vez es deseable que el programa no sea interrumpido constantemente, dado que tal vez se necesiten condiciones muy simples de análisis, como puede ser un umbral. En este ejemplo, se demuestra la potencia del periférico *ADC* al utilizar el *threshold compare*. El mismo es configurado para generar interrupciones cuando el valor convertido en el *Canal 0* cruce por ciertos umbrales estipulados por el usuario. Para el ejemplo, dependiendo de si la conversión que generó la interrupción estaba entre los valores de umbral, o por fuera, se enciende alguno de los LEDs RGB (rojo o azul) del stick de desarrollo, sin una interrupción constante por parte de la finalización de conversión de la *secuencia A*.

Posibilidad de interrupcion de secuencia de conversión A

La *secuencia A* está configurada como *baja prioridad*. Esto implica que cualquier disparo de conversión de *secuencia B* frenará las conversiones de la *secuencia A*, realizará la/s conversiones de la *secuencia B* y luego retomará las conversiones de *secuencia A*. Como la *secuencia A* está configurada para convertir continuamente, queda en real evidencia la condición de baja prioridad de la *secuencia A*.

```
/**
 * @file Ejemplo_ADC.c
 * @brief Ejemplo de utilización del \e ADC con la librería (capa de aplicación)
 *
 * # Configuraciones
 *
 * El programa utiliza el clock por default con el que comienza el microcontrolador, es decir, el <em>Free
      Running
 * Oscillator</em> funcionando a 12MHz
 *
 * El periférico \e ADC será configurado con las siguientes características:
 *  - Funcionamiento \b sincrónico
 *  - Frecuencia de muestreo de 1Mhz
 *  - Modo bajo consumo inhabilitado
 *  .
 *
 * La secuencia A es configurada para generar conversiones en el canal 0:
 *  - El canal 0 está conectado al preset propio del stick de desarrollo (Puerto 0 pin 7)
 *  .
 *
 * La secuencia B es configurada para generar conversiones en los canales 4 y 8:
 *  - El canal 4 está ubicado en el pin número 7 (Puerto 0 pin 22)
 *  - El canal 8 está ubicado en el pin número 3 (Puerto 0 pin 18)
 *  .
 *
 * En ambos canales de la secuencia B se puede conectar un preset con los terminales de los extremos uno a
      VDD y el
 * otro a VSS, y el terminal central a cada uno de los canales.
 *
 * Además, la secuencia A tendrá la siguiente configuración:
 *  - Trigger: Únicamente se disparan conversiones por software
 *  - Bypass sincronismo: Sí
 *  - Modo de interrupción: Cuando termina cada canal
 *  - Burst: Habilitado
 *  - Un trigger dispara: Una conversión de canal
 *  - Secuencia A como baja prioridad: Si
 *  .
 *
 * La secuencia B tendrá la siguiente configuración:
 *  - Trigger: Únicamente se disparan conversiones por software
 *  - Bypass sincronismo: Sí
 *  - Modo de interrupción: Cuando se termina la secuencia completa
 *  - Burst: Inhabilitado
 *  - Un trigger dispara: Una conversión de secuencia completa
 *  .
 *
 * Una vez inicializado el periférico, se configura el periférico \e Systick para interrumpir cada 1
      milisegundo
 * y mediante su manejador se lleva la cuenta de los milisegundos transcurridos. Una vez transcurridos
 * 1 segundo, se dispara una conversión de \e ADC, y sus resultados se guardan en dos variables globales.
 *
 * # Descripción de funcionamiento
 *
 * La idea de este ejemplo es demostrar las capacidades de la librería para con el periférico \e ADC. Con
      dicho
 * fin, se muestran las dos principales capacidades del periférico, incluidas particularidades explicadas a
 * continuación.
 *
 * ## Conversiones no continuas
 *
 * La <em>secuencia B</em> está configurada para generar conversiones disparadas por software. Esta forma es
 * usualmente utilizada cuando no es necesario tener conversiones continuas, dada la naturaleza de necesidad
      de
 * la aplicación a desarrollar. Sin embargo, no es la óptima para distintos casos como pueden ser, por
      ejemplo:
 *      - Conversiones con tiempos precisos. En estos casos, se recomienda disparar conversiones disparadas
 *      por algún timer, la librería tiene implementada esta configuración.
 *      - Conversiones que dependen de variables externas. En estos casos, es útil disparar las conversiones
 *      mediante interrupciones de \e PININT o \e ACOMP.
 *      .
 *
 * En el caso del ejemplo, se disparan conversiones cada aproximadamente 100 milisegundos con ayuda del \e
      Systick.
 *
 * ## Conversiones continuas
 *
 * La <em>secuencia A</em>, al tener el modo \e BURST habilitado, genera conversiones continuamente.
      Dependiendo
```

```
 * de la necesidad de procesamiento de dichos resultados, tal vez es deseable que el programa no sea
      interrumpido
 * constantemente, dado que tal vez se necesiten condiciones muy simples de análisis, como puede ser un
      umbral. En
 * este ejemplo, se demuestra la potencia del periférico \e ADC al utilizar el <em>threshold compare</em>.
      El
 * mismo es configurado para generar interrupciones cuando el valor convertido en el <em>Canal 0</em> cruce
      por
 * ciertos umbrales estipulados por el usuario. Para el ejemplo, dependiendo de si la conversión que generó
      la
 * interrupción estaba entre los valores de umbral, o por fuera, se enciende alguno de los LEDs RGB (rojo o
      azul)
 * del stick de desarrollo, sin una interrupción constante por parte de la finalización de conversión de la
 * <em>secuencia A</em>.
 *
 * ## Posibilidad de interrupcion de secuencia de conversión A
 *
 * La <em>secuencia A</em> está configurada como <em>baja prioridad</em>. Esto implica que cualquier disparo
      de
 * conversión de <em>secuencia B</em> frenará las conversiones de la <em>secuencia A</em>, realizará la/s
 * conversiones de la <em>secuencia B</em> y luego retomará las conversiones de <em>secuencia A</em>. Como
      la
 * <em>secuencia A</em> está configurada para convertir continuamente, queda en real evidencia la condición
      de
 * baja prioridad de la <em>secuencia A</em>.
 *
 * @author Augusto Santini
 * @author Esteban E. Chiama
 * @date 4/2020
 */
#include <cr_section_macros.h>
#include <stddef.h>
#include <HAL_GPIO.h>
#include <HAL_ADC.h>
#include <HAL_SYSTICK.h>

/** Máscara de configuración de canales habilitados para la secuencia A */
#define     ADC_SEQA_CHANNELS           (1 « 0)

/** Canal donde se encuentra el preset del stick de desarrollo */
#define     ADC_PRESET_CHANNEL          (0)

/** Máscara de configuración de canales habilitados para la secuencia B */
#define     ADC_SEQB_CHANNELS           ((1 « 4) | (1 « 8))

/** 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)

/** Tiempo de disparo de conversiones de \e ADC en \b milisegundos */
#define     ADC_CONVERSION_TIME_MSEG    (100)

/** Valor de umbral bajo para las comparaciones del ADC */
#define     THR_LOW                     (1000)

/** Valor de umbral alto para las comparaciones del ADC */
#define     THR_HIGH                    (2500)

/** Puerto de ambos LEDS */
#define     LED_PORT                    (1)

/** Puerto y pin de Led Azul */
#define     LED_AZUL                    (HAL_GPIO_PORTPIN_1_1)

/** Puerto y pin de Led Rojo */
#define     LED_ROJO                    (HAL_GPIO_PORTPIN_1_2)

/** Valor lógico de LED encendido */
#define     LED_ON_STATE                (0)

/** Valor lógico de LED apagado */
#define     LED_OFF_STATE               (1)
static void adc_sequence_callback(void);
static void adc_thr_callback(void);
static void systick_callback(void);

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

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

/** Variable para guardar el resultado de una comparación válida contra el umbral */
static hal_adc_channel_compare_result_t comparison_result =
{
```

```
    .value = 0,
    .result_range = 0,
    .result_crossing = 0
};

/** Configuración de la secuencia A. Como no va a cambiar es declarada \e const */
static const hal_adc_sequence_config_t sequence_a_config =
{
    .channels = ADC_SEQA_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_EOC,
    .burst = 1,
    .single_step = 1,
    .low_priority = 1,
    .callback = NULL
};

/** Configuración de la secuencia B. Como no va a cambiar es declarada \e const */
static const hal_adc_sequence_config_t sequence_b_config =
{
    .channels = ADC_SEQB_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_sequence_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_sequence_config(HAL_ADC_SEQUENCE_SEL_A, &sequence_a_config);
    hal_adc_sequence_config(HAL_ADC_SEQUENCE_SEL_B, &sequence_b_config);
    // Configuración de comparaciones e interrupción
    hal_adc_threshold_config(HAL_ADC_THRESHOLD_SEL_0, THR_LOW, THR_HIGH);
    hal_adc_threshold_channel_config(ADC_PRESET_CHANNEL, HAL_ADC_THRESHOLD_SEL_0,
       HAL_ADC_THRESHOLD_IRQ_SEL_CROSSING);
    hal_adc_threshold_register_interrupt(adc_thr_callback);
    // Inicialización del \e Systick con el tiempo de tick adecuado
    hal_systick_init(TICK_TIME_USEG, systick_callback);
    // Inicialización de GPIO correspondiente a LEDs
    hal_gpio_init(LED_PORT);
    hal_gpio_set_dir(LED_AZUL, HAL_GPIO_DIR_OUTPUT, LED_OFF_STATE);
    hal_gpio_set_dir(LED_ROJO, HAL_GPIO_DIR_OUTPUT, LED_OFF_STATE);
    // Comienzo de conversiones (modo burst) en secuencia A
    hal_adc_sequence_start(HAL_ADC_SEQUENCE_SEL_A);
    while(1)
    {
        if(flag_secuencia_adc_completada == 1)
        {
            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)
    {
        hal_adc_sequence_start(HAL_ADC_SEQUENCE_SEL_B);
    }
}

/**
 * @brief Callback a ejecutar en cada finalización de conversión de \b secuencia de \e ADC
 */
static void adc_sequence_callback(void)
```

```
{
    // Obtención de resultados de conversión
    hal_adc_sequence_get_result(HAL_ADC_SEQUENCE_SEL_B, resultados_conversion_adc);
    flag_secuencia_adc_completada = 1;
}

/**
 * @brief Callback cuando los canales habilitados cumplen con la condicion de umbral establecida
 */
static void adc_thr_callback(void)
{
    hal_adc_threshold_get_comparison_results(&comparison_result);
    // Chequeo si la interrupción se disparó porque la conversión quedó por dentro o fuera del rango
    if(comparison_result.result_range == HAL_ADC_COMPARISON_RANGE_INSIDE)
    {
        hal_gpio_set_pin(LED_ROJO);     // Apaga rojo
        hal_gpio_clear_pin(LED_AZUL); // Prende azul
    }
    else
    {
        hal_gpio_set_pin(LED_AZUL);     // Apaga azul
        hal_gpio_clear_pin(LED_ROJO); // Prende rojo
    }
}
```

5.2. Ejemplo_DAC.c

El programa utiliza el *DAC* para generar una señal tipo rampa en la salida del canal 0 (Puerto 0 Pin 17) el cual está mapeado al pin número 2 del stick de desarrollo. Ver Acerca del stick de desarrollo LPC845_BRK para más información.

Para poder tener una visualización del programa de ejemplo, sera necesario un osciloscopio para mayor presición, aunque también se puede utilizar un LED con un resistor en serie.

- En caso de utilizar un osciloscopio: Conectar la punta de medición al pin 2 del stick de desarrollo. Debería observar una señal tipo rampa, con un período de aproximadamente un segundo y tensión pico aproximadamente 3.3V.
- En caso de utilizar un LED con un resistor en serie: Conectar un extremo del resistor al pin 2 del stick de desarrollo, el otro extremo del resistor conectarlo con el ánodo del LED, y el cátodo del LED conectarlo a la masa del stick de desarrollo. Debería ver como el LED enciende gradualmente hasta apagarse y volver a repetir el ciclo.

El periférico *Systick* se configura para generar interrupciones cada 1 milisegundo. En cada tick, se escribe en el *DAC* un valor de una tabla, cuyo indice ira incrementando en cada tick. En la tabla se encuentran guardados los valores de la señal tipo rampa, incializados al comienzo del programa en un lazo *for*.

El periférico *DAC* es inicializado con un tiempo de asentamiento de 1 microsegundo.

```
/**
 * @file Ejemplo_DAC.c
 * @brief Ejemplo de utilización del \e DAC con la librería (capa de aplicación)
 *
 * El programa utiliza el \e DAC para generar una señal tipo rampa en la salida del canal 0 (Puerto 0 Pin
      17) el
 * cual está mapeado al pin número 2 del stick de desarrollo. Ver @ref acerca_del_stick para más
      información.
 *
 * Para poder tener una visualización del programa de ejemplo, sera necesario un osciloscopio para mayor
      presición,
 * aunque también se puede utilizar un LED con un resistor en serie.
 *
 * - En caso de utilizar un osciloscopio: Conectar la punta de medición al pin 2 del stick de desarrollo.
      Debería
 * observar una señal tipo rampa, con un período de aproximadamente un segundo y tensión pico
      aproximadamente 3.3V.
 * - En caso de utilizar un LED con un resistor en serie: Conectar un extremo del resistor al pin 2 del
      stick de
 * desarrollo, el otro extremo del resistor conectarlo con el ánodo del LED, y el cátodo del LED conectarlo
      a la
```

```
 * masa del stick de desarrollo. Debería ver como el LED enciende gradualmente hasta apagarse y volver a
       repetir
 * el ciclo.
 *
 * El periférico \e Systick se configura para generar interrupciones cada 1 milisegundo. En cada tick, se
       escribe en
 * el \e DAC un valor de una tabla, cuyo indice ira incrementando en cada tick. En la tabla se encuentran
       guardados
 * los valores de la señal tipo rampa, incializados al comienzo del programa en un lazo \e for.
 *
 * El periférico \e DAC es inicializado con un tiempo de asentamiento de 1 microsegundo.
 *
 * @author Augusto Santini
 * @date 4/2020
 */
#include <cr_section_macros.h>
#include <HAL_DAC.h>
#include <HAL_SYSTICK.h>

/** Tiempo de interrupción del \e Systick en \b microsegundos */
#define     TICK_TIME_USEG              (1000)

/** Canal de \e DAC a utilizar en el programa */
#define     CANAL_DAC                   (HAL_DAC_0)

/** Cantidad de muestras definidas para la tabla de muestreo */
#define     CANTIDAD_MUESTRAS       (1000)

/** Máximo valor que se puede escribir en el \e DAC */
#define     VALOR_MAXIMO_DAC        ((1 << 10) - 1)
static void tick_callback(void);

/** Tabla a inicializar al comienzo del programa */
static uint16_t tabla_rampa[CANTIDAD_MUESTRAS];
int main(void)
{
    uint32_t contador;
    // Definicion de los valores para la señal tipo rampa
    for(contador = 0; contador < CANTIDAD_MUESTRAS; contador++)
    {
        tabla_rampa[contador] = contador * (VALOR_MAXIMO_DAC / CANTIDAD_MUESTRAS);
    }
    // Inicializacion del \e DAC
    hal_dac_init(CANAL_DAC, HAL_DAC_SETTLING_TIME_1US_MAX, 0);
    // Inicialización del \e Systick con el tiempo de tick adecuado
    hal_systick_init(TICK_TIME_USEG, tick_callback);
    while(1)
    {
    }
    return 0;
}

/**
 * @brief Callback a ejecutar en cada tick del \e Systick
 */
static void tick_callback(void)
{
    static uint32_t contador = 0;
    hal_dac_update_value(CANAL_DAC, tabla_rampa[contador++]);
    contador %= CANTIDAD_MUESTRAS;
}
```

5.3. Ejemplo_GPIO.c

Configuraciones

El programa utiliza únicamente el LED RGB y el pulsador de usuario del stick de desarrollo.

- LED RGB - Rojo: Puerto 1, Pin 2
- LED RGB - Azul: Puerto 1, Pin 1
- Pulsador de usuario: Puerto 0, Pin 4

Ver Acerca del stick de desarrollo LPC845_BRK para más información.

Se configura el *Systick* para generar ticks cada 1 milisegundo.

Se configura el pin del pulsador para utilizar lógica invertida en su hardware.

Funcionamiento del programa

El *Systick* se utiliza para generar un titileo en el LED RGB *rojo* siempre y cuando el pulsador de usuario no esté presionado. Mientras el pulsador de usuario se encuentre presionado, el LED RGB *azul* estará fijo y el *rojo* quedará apagado. El titileo nunca deja de suceder, sino que se enmascara la salida correspondiente. Esto implica que si el pulsador se deja de presionar en el momento en que el mismo tiene el LED RGB *rojo* encendido, no se va a ver el LED RGB *azul* encender, dado el tipo de conexión en el stick de desarrollo y los valores de tensión de LED.

```
/**
 * @file Ejemplo_GPIO.c
 * @brief Ejemplo de utilización del \e GPIO con la librería (capa de aplicación)
 *
 * # Configuraciones
 *
 * El programa utiliza únicamente el LED RGB y el pulsador de usuario del stick de desarrollo.
 * - LED RGB - Rojo: Puerto 1, Pin 2
 * - LED RGB - Azul: Puerto 1, Pin 1
 * - Pulsador de usuario: Puerto 0, Pin 4
 *
 * Ver @ref acerca_del_stick para más información.
 *
 * Se configura el \e Systick para generar ticks cada 1 milisegundo.
 *
 * Se configura el pin del pulsador para utilizar lógica invertida en su hardware.
 *
 * # Funcionamiento del programa
 *
 * El \e Systick se utiliza para generar un titileo en el LED RGB \e rojo siempre y cuando el pulsador de
      usuario
 * no esté presionado. Mientras el pulsador de usuario se encuentre presionado, el LED RGB \e azul estará
      fijo y
 * el \e rojo quedará apagado. El titileo nunca deja de suceder, sino que se enmascara la salida
      correspondiente.
 * Esto implica que si el pulsador se deja de presionar en el momento en que el mismo tiene el LED RGB \e
      rojo
 * encendido, no se va a ver el LED RGB \e azul encender, dado el tipo de conexión en el stick de desarrollo
 * y los valores de tensión de LED.
 *
 * @author Augusto Santini
 * @date 4/2020
 */
#include <cr_section_macros.h>
#include <HAL_SYSTICK.h>
#include <HAL_GPIO.h>
#include <HAL_IOCON.h>

/** Tiempo de interrupción del \e Systick en \b microsegundos */
#define     TICK_TIME_USEG              (1000)

/** Puerto/pin del LED RGB \e rojo */
#define     LED_RGB_RED_PORTPIN         (HAL_GPIO_PORTPIN_1_2)

/** Puerto/pin del LED RGB \e azul */
#define     LED_RGB_BLUE_PORTPIN        (HAL_GPIO_PORTPIN_1_1)

/** Puerto/pin del switch */
#define     KEY_PORTPINT                (HAL_GPIO_PORTPIN_0_4)

/** Tiempo de titileo en \e milisegundos */
#define     BLINK_TIME_MSEG             (1000)

/** Valor que apaga el LED RGB, sea el azul o el rojo */
#define     LED_RGB_OFF                 (1)

/** Valor que enciende el LED RGB, sea el azul o el rojo */
#define     LED_RGB_ON                  (0)

/** Macro para apagar el LED RGB \e azul */
#define     LED_RGB_BLUE_OFF            (hal_gpio_set_pin(LED_RGB_BLUE_PORTPIN))

/** Macro para encender el LED RGB \e azul */
#define     LED_RGB_BLUE_ON             (hal_gpio_clear_pin(LED_RGB_BLUE_PORTPIN))
static void tick_callback(void);

/** Configuración de hardware del pin del pulsador de usuario */
static const hal_iocon_config_t key_config =
{
    .pull_mode = HAL_IOCON_PULL_NONE,
    .hysteresis = 0,
    .invert_input = 1,
    .open_drain = 0,
    .sample_mode = HAL_IOCON_SAMPLE_MODE_BYPASS,
    .clk_sel = HAL_IOCON_CLK_DIV_0,
    .iic_mode = HAL_IOCON_IIC_MODE_GPIO
};
```

```
/**
 * @brief Punto de entrada del programa
 * @return Nunca deberia terminar esta función
 */
int main(void)
{
    // Inicialización de ambos puertos de GPIO
    // En este caso se necesitan los dos, dado que el switch esta en el puerto 0 y los LED RGB en el puerto
       1
    hal_gpio_init(HAL_GPIO_PORT_0);
    hal_gpio_init(HAL_GPIO_PORT_1);
    // Inicialización de los pines de los LEDs RGB como salida y con los LEDs apagados
    hal_gpio_set_dir(LED_RGB_RED_PORTPIN, HAL_GPIO_DIR_OUTPUT, LED_RGB_OFF);
    hal_gpio_set_dir(LED_RGB_BLUE_PORTPIN, HAL_GPIO_DIR_OUTPUT, LED_RGB_OFF);
    // Configuración del pin del switch para utilizar lógica invertida
    hal_iocon_config_io(KEY_PORTPINT, &key_config);
    // Inicialización del pin del switch como entrada
    hal_gpio_set_dir(KEY_PORTPINT, HAL_GPIO_DIR_INPUT, 0);
    // Inicialización del \e Systick con el tiempo de tick adecuado
    hal_systick_init(TICK_TIME_USEG, tick_callback);
    while(1)
    {
        /*
         * Si bien el pulsador lleva el nivel físico del pin a VSS al ser presionado, se configuró el mismo
       para
         * utilizar lógica invertida, es decir que en este caso, al leer 0 se está leyendo que el pulsador
       NO
         * está siendo presionado.
         */
        if(hal_gpio_read_pin(KEY_PORTPINT) == 0)
        {
            // El switch no está presionado
            hal_gpio_set_mask_bits(HAL_GPIO_PORTPIN_TO_PORT(LED_RGB_RED_PORTPIN), 1 «
       HAL_GPIO_PORTPIN_TO_PIN(LED_RGB_RED_PORTPIN));
            LED_RGB_BLUE_ON;
        }
        else
        {
            // El switch está presionado
            hal_gpio_clear_mask_bits(HAL_GPIO_PORTPIN_TO_PORT(LED_RGB_RED_PORTPIN), 1 «
       HAL_GPIO_PORTPIN_TO_PIN(LED_RGB_RED_PORTPIN));
            LED_RGB_BLUE_OFF;
        }
        /*
         * NOTA: El LED RGB \e verde y la máscara necesaria están siendo escritas constantemente. Esto no es
       ideal,
         * lo ideal sería identificar en qué momento el switch cambia de estado y escribir una única vez, en
       el
         * momento que sea necesario.
         */
    }
    return 0;
}

/**
 * @brief Callback a ejecutar en cada tick
 */
static void tick_callback(void)
{
    static uint32_t contador = 0;
    contador = (contador + 1) % BLINK_TIME_MSEG;
    if(!contador)
    {
        // Titileo cada 1 segundo
        hal_gpio_masked_toggle_port(HAL_GPIO_PORTPIN_TO_PORT(LED_RGB_RED_PORTPIN), 1 «
       HAL_GPIO_PORTPIN_TO_PIN(LED_RGB_RED_PORTPIN));
    }
}
```

5.4. Ejemplo_PININT.c

Configuraciones

El programa utiliza el clock por default con el que comienza el microcontrolador, es decir, el *Free Running Oscillator* funcionando a 12MHz

El periférico *PININT* será configurado de la siguiente manera:

- Canal 0 en el pulsador de usuario (Puerto 0 ; Pin 4)
- Detecciones por nivel (inicialmente nivel bajo, ver descripción de programa)

Se configura el pin correspondiente al LED RGB *Rojo* como salida.

```
/**
 * @file Ejemplo_PININT.c
 * @brief Ejemplo de utilización del \e PININT con la librería (capa de aplicación)
 *
 * # Configuraciones
 *
 * El programa utiliza el clock por default con el que comienza el microcontrolador, es decir, el <em>Free
      Running
 * Oscillator</em> funcionando a 12MHz
 *
 * El periférico \e PININT será configurado de la siguiente manera:
 *      - Canal 0 en el pulsador de usuario (Puerto 0 ; Pin 4)
 *      - Detecciones por nivel (inicialmente nivel bajo, ver descripción de programa)
 *      .
 *
 * Se configura el pin correspondiente al LED RGB \e Rojo como salida.
 *
 * # Descripción del programa
 *
 * Cuando el usuario accione el pulsador de usuario, se detectará dicho evento mediante el periférico @ref
      PININT,
 * ejecutando el callback correspondiente. En el mismo, se inhabilitarán las detecciones por <em>nivel
      bajo</em>
 * en dicho puerto/pin, se encenderá el LED RGB \e Rojo y se disparará el @ref WKT con un tiempo de
 * aproximadamente dos segundos. Una vez transcurrido el tiempo, se volverán a habilitar las detecciones por
 * <em>nivel bajo</em>, y se apagará el LED RBG \e Rojo.
 *
 * Si el usuario mantiene presionado el pulsador de usuario, verá el LED RGB \e Rojo siempre encendido, dado
      que
 * al volver a habilitar las detecciones, inmediatamente disparará el callback de detección.
 *
 * @author Augusto Santini
 * @date 4/2020
 */
#include <cr_section_macros.h>
#include <stddef.h>
#include <HAL_PININT.h>
#include <HAL_GPIO.h>
#include <HAL_IOCON.h>
#include <HAL_WKT.h>

/** Tiempo de inhibición del pulsador una vez detectado un cambio */
#define     KEY_INHIBIT_USEG            (2e6)

/** Puerto y pin de LED ROJO */
#define     LED_ROJO                    (HAL_GPIO_PORTPIN_1_2)

/** Macro para encender el LED ROJO */
#define     LED_ROJO_ON()               (hal_gpio_clear_pin(LED_ROJO))

/** Macro para apagar el LED ROJO */
#define     LED_ROJO_OFF()              (hal_gpio_set_pin(LED_ROJO))

/** Puerto/pin del switch */
#define     KEY_PORTPINT                (HAL_GPIO_PORTPIN_0_4)

/** Canal del \e PININT para el pulsador de usuario */
#define     KEY_PININT_CHANNEL          (HAL_PININT_CHANNEL_0)
static void pinint_callback(void);
static void wkt_callback(void);
int main(void)
{
    // Inicialización de GPIO
    hal_gpio_init(HAL_GPIO_PORT_0);
    hal_gpio_init(HAL_GPIO_PORT_1);
    hal_gpio_set_dir(LED_ROJO, HAL_GPIO_DIR_OUTPUT, 0);
    LED_ROJO_OFF();
    // Configuración de PININT
    hal_pinint_init();
    hal_pinint_channel_config(KEY_PININT_CHANNEL, KEY_PORTPINT, pinint_callback);
    hal_pinint_level_detections_config(KEY_PININT_CHANNEL, HAL_PININT_LEVEL_DETECTIONS_LOW);
    // Inicialización del WKT
    hal_wkt_init(HAL_WKT_CLOCK_SOURCE_FRO_DIV, 0, wkt_callback);
    while(1)
    {
    }
    return 0;
}

/**
```

```
 * @brief Callback a ejecutar cuando se detecte el evento configurado
 */
static void pinint_callback(void)
{
    LED_ROJO_ON();
    // Inhibición de interrupción de PININT
    hal_pinint_level_detections_config(KEY_PININT_CHANNEL, HAL_PININT_LEVEL_DETECTIONS_NONE);
    hal_wkt_start_count(KEY_INHIBIT_USEG);
}

/**
 * @brief Callback a ejecutar cuando termina el conteo del \e WKT
 */
static void wkt_callback(void)
{
    LED_ROJO_OFF();
    // Re-activación de interrupción de PININT
    hal_pinint_level_detections_config(KEY_PININT_CHANNEL, HAL_PININT_LEVEL_DETECTIONS_LOW);
}
```

5.5. Ejemplo_SYSCON.c

El programa configura el FRO para funcionar sin divisor y con la frecuencia configurada por default, obteniendo una frecuencia de funcionamiento del mismo de 24MHz. Además, selecciona el FRO como clock principal de sistema.

Luego configura la salida CLKOUT para salir en el Puerto 0 ; Pin 18 mediante un divisor por 24, obteniendo una frecuencia de salida en el pin Puerto 0 ; Pin 18 de 1MHz.

La única forma de medir dicha salida es mediante un osciloscopio.

```
/**
 * @file Ejemplo_SYSCON.c
 * @brief Ejemplo de utilización del \e SYSCON con la librería (capa de aplicación)
 *
 * El programa configura el FRO para funcionar sin divisor y con la frecuencia configurada por default,
      obteniendo
 * una frecuencia de funcionamiento del mismo de 24MHz. Además, selecciona el FRO como clock principal de
      sistema.
 *
 * Luego configura la salida CLKOUT para salir en el Puerto 0 ; Pin 18 mediante un divisor por 24,
      obteniendo una
 * frecuencia de salida en el pin Puerto 0 ; Pin 18 de 1MHz.
 *
 * La única forma de medir dicha salida es mediante un osciloscopio.
 *
 * @author Augusto Santini
 * @date 4/2020
 */
#include <cr_section_macros.h>
#include <HAL_SYSCON.h>
#include <HAL_GPIO.h>

/** Puerto/pin de salida para el CLKOUT */
#define     CLKOUT_PORTPIN      (HAL_GPIO_PORTPIN_0_18)

/** Divisor para el CLKOUT */
#define     CLKOUT_DIVISOR      (24)

/**
 * @brief Punto de entrada del programa
 * @return Nunca deberia terminar esta función
 */
int main(void)
{
    // Configuración del FRO para que esté sin divisor
    hal_syscon_fro_clock_config(1);
    // Selección del FRO como clock principal
    hal_syscon_system_clock_set_source(HAL_SYSCON_SYSTEM_CLOCK_SEL_FRO);
    // Clock principal en un pin (utilizando un divisor)
    hal_syscon_clkout_config(CLKOUT_PORTPIN, HAL_SYSCON_CLKOUT_SOURCE_SEL_MAIN_CLOCK, CLKOUT_DIVISOR);
    while(1)
    {
    }
    return 0;
}
```

5.6. Ejemplo_WKT.c

Configuraciones

El programa utiliza la configuración del clock por default, es decir, FRO como fuente de clock principal, y el mismo con una frecuencia de 12MHz.

El periférico *WKT* es configurado para utilizar como fuente de clock el oscilador de bajo consumo, o el FRO dividido, dependiendo de un define ubicado en el código. De esta forma, se puede probar la presición de ambas fuentes de clock, y se puede observar la gran incertidumbre en el clock de bajo consumo.

Funcionamiento del programa

En el callback de interrupción del *WKT* configurado, se genera una inversión en el estado del pin del LED RGB *verde* y se vuelve a disparar el conteo. Esto se realiza cada aproximadamente 1 milisegundo.

Nótese que la carga del valor de conteo es realizada mediante la función hal_wkt_start_count_with_value y no con hal_wkt_start_count. Esto se debe a que el tiempo que queremos disparar es conocido, y es un tiempo relativamente corto para las frecuencias de clock que se utilizan. Para más información, referirise a Wake Up Timer (WKT).

```
/**
 * @file Ejemplo_WKT.c
 * @brief Ejemplo de utilización del \e WKT con la librería (capa de aplicación)
 *
 * # Configuraciones
 *
 * El programa utiliza la configuración del clock por default, es decir, FRO como fuente de clock principal,
 * y el mismo con una frecuencia de 12MHz.
 *
 * El periférico \e WKT es configurado para utilizar como fuente de clock el oscilador de bajo consumo, o el
 * FRO dividido, dependiendo de un define ubicado en el código. De esta forma, se puede probar la presición
      de
 * ambas fuentes de clock, y se puede observar la gran incertidumbre en el clock de bajo consumo.
 *
 * # Funcionamiento del programa
 *
 * En el callback de interrupción del \e WKT configurado, se genera una inversión en el estado del pin del
      LED
 * RGB \e verde y se vuelve a disparar el conteo. Esto se realiza cada aproximadamente 1 milisegundo.
 *
 * Nótese que la carga del valor de conteo es realizada mediante la función @ref
      hal_wkt_start_count_with_value
 * y no con @ref hal_wkt_start_count. Esto se debe a que el tiempo que queremos disparar es conocido, y es
      un
 * tiempo relativamente corto para las frecuencias de clock que se utilizan. Para más información,
      referirise a
 * @ref WKT.
 *
 * @author Augusto Santini
 * @date 4/2020
 */
#include <cr_section_macros.h>
#include <HAL_WKT.h>
#include <HAL_GPIO.h>

/** Valor de tick para el \e WKT en microsegundos */
#define     WKT_TICK_USEG               (1000)

/** Puerto/pin del LED RGB \e verde */
#define     LED_RGB_GREEN_PORTPIN       (HAL_GPIO_PORTPIN_1_0)

/** Si este define queda sin comentar, se utilizará el oscilador de bajo consumo como fuente de clock del \e
      WKT */
#define     OSC_LOW_POWER
static void wkt_callback(void);
#ifdef      OSC_LOW_POWER
static const uint32_t wkt_reload_val = 10;
#else
static const uint32_t wkt_reload_val = 750;
#endif
int main(void)
{
    // Inicialización de GPIO
    hal_gpio_init(HAL_GPIO_PORT_1);
```

```
    hal_gpio_set_dir(LED_RGB_GREEN_PORTPIN, HAL_GPIO_DIR_OUTPUT, 0);
    // Inicialización del WKT
#ifdef      OSC_LOW_POWER
    hal_wkt_init(HAL_WKT_CLOCK_SOURCE_LOW_POWER_OSC, 0, wkt_callback);
#else
    hal_wkt_init(HAL_WKT_CLOCK_SOURCE_FRO_DIV, 0, wkt_callback);
#endif
    hal_wkt_start_count_with_value(wkt_reload_val);
    while(1)
    {
    }
    return 0;
}

/**
 * @brief Callback a ejecutar en interrupción del WKT
 */
static void wkt_callback(void)
{
    // Toggle del LED RGB verde
    hal_gpio_toggle_pin(LED_RGB_GREEN_PORTPIN);
    // Re-disparo del WKT
    hal_wkt_start_count_with_value(wkt_reload_val);
}
```

Índice alfabético