API de la biblioteca sAPI

Módulos

sAPI_DataTypes

Define las siguientes constantes:

Estados lógicos

```
#define FALSE 0
```

```
#define TRUE !FALSE
```

Estados funcionales

```
#define ON 1
```

```
#define OFF 0
```

Estados eléctricos

```
#define HIGH 1
```

```
#define LOW 0
```

Además define los tipos de datos:

- **Booleano** `bool_t`
- **Enteros sin signo** `uint8_t, uint16_t, uint32_t, uint64_t`
- **Enteros con signo** `int8_t, int16_t, int32_t, int64_t`

El tipo de datos para el conteo de tiempo en la unidad Tick

```
typedef uint64_t tick_t;
```

Un tipo de datos para puntero a función:

```
typedef bool_t (*sAPI_FuncPtr_t)(void *);
```

- Parámetro: `void *` Para poder pasar cualquier argumento.
- Retorna: `bool_t` Para reportar errores (TRUE si todo está bien).

Utilizando este tipo de datos define la finción Nula que no hace nada y Retorna siempre TRUE, esta se utiliza para evitar errores de NULL POINTER.

```
bool_t sAPI_NullFuncPtr(void *);
```

- Parámetro: `void *` No usado.
- Retorna: `bool_t` Retorna siempre TRUE.

sAPI_PeripheralMap

Contiene el mapa de periféricos.

DigitalIO Map

EDU-CIAA-NXP:

```
DIO0,  DIO1,  DIO2,  DIO3,  DIO4,  DIO5,  DIO6,  DIO7,
DIO8,  DOI9,  DIO10, DIO11, DIO12, DIO13, DIO14, DIO15,
DIO16, DIO17, DIO18, DIO19, DIO20, DIO21, DIO22, DIO23,
DIO24, DIO25, DIO26, DIO27, DIO28, DIO29, DIO30, DIO31,
DIO32, DIO33, DIO34, DIO35,
TEC1,  TEC2,  TEC3,  TEC4,
LED1,  LED2,  LED3,  LEDR,  LEDG,  LEDB
```

CIAA-NXP:

```
DI0,   DI1,   DI2,   DI3,   DI4,   DI5,   DI6,   DI7,
DO0,   DO1,   DO2,   DO3,   DO4,   DO5,   DO6,   DO7
```

AnalogIO Map

EDU-CIAA-NXP: `AI0, AI1, AI2, AO`

CIAA-NXP: `AI0, AI1, AI2, AI3, AO`

Uart Map

```
UART_USB, UART_232, UART_485
```

Pwm Map

```
PWM0, PWM1, PWM2, PWM3, PWM4, PWM5, PWM6, PWM7, PWM8, PWM9, PWM10
```

Servo Map

```
SERVO0, SERVO1, SERVO2, SERVO3, SERVO4, SERVO5, SERVO6, SERVO7, SERVO8
```

I2C Map

`I2C0`

sAPI_IsrVector

Contiene la tabla de vectores de interrupción.

sAPI_Board

Contiene la función de configuración para inicialización de la plataforma de hardware:

```
void boardConfig( void );
```

- Parámetro: `void`
- Retorna: `void`

sAPI_Tick

Configuración de interrupción periódica

```
bool_t tickConfig(tick_t tickRateMSvalue, sAPI_FuncPtr_t tickHook );
```

- Parámetro: `tick_t tickRateMSvalue` cada cuantos ms ocurre un tick.
- Parámetro: `sAPI_FuncPtr_t tickHook` función a ejecutar en cada tick.
- Retorna: `bool_t` TRUE en en caso correcto o FALSE en caso de errores.

Configura una interrupción periódica de temporizador cada tickRateMSvalue milisegundos para utilizar de base de tiempo del sistema. Una vez ejecutada esta función se dice que ocurre un tick del sistema cada tickRateMSvalue milisegundos.

La tasa de ticks en ms, tickRateMS, es un parámetro con rango de 1 a 50 ms.

Además de aumentar el conteo de ticks en cada interrupción, la función tickConfig ejecuta la función pasada como parámero cada vez que ocurre un tick. Si no se desea ejecutar ninguna función debe poner en cero este parámetro.

Leer la variable del conteo actual de ticks

```
tick_t tickRead( void );
```

- Parámetro: `void` sin parámetro.
- Retorna: `tick_t` el valor actual del contador de ticks.

La variable del conteo actual de ticks se incrementa en 1 cada tickRateMSvalue milisegundos.

Escribir la variable del conteo actual de ticks

`void tickWrite( tick_t ticks );`

- Parámetro: `tick_t ticks` el nuevo valor a setear del contador de ticks.
- Retorna: `void`

Se utiliza si se necesita cambiar el valor del contador de ticks, por ejemplo, para resetearlo.

En la implementación para la CIAA utiliza internaente el peiférico temporizador Systick para configurar una interrupción periódica.

sAPI_Delay

Para utilizar los retardos (con excepción del retardo inexacto) se debe configurar el Tick ya que utiliza estas interrupciones como base de tiempo.

Todos los tiempos de parámetros están en milisegundos.

Define la constante `#define INACCURATE_TO_MS 20400` y contiene las funciones:

Retardo inexacto bloqueante `void delayInaccurate(tick_t delay_ms);`

- Parámetros: `tick_t delay_ms` tiempo de duración del retardo en ms.
- Retorna: `void`

Utiliza un bloque for bloqueante que tiene una constante calculada "a ojo" (INACCURATE_TO_MS) para perder muchos ciclos de reloj y lograr hacer un retado.

Retardo bloqueante `void delay (tick_t time);`

- Parámetros: `tick_t time`
- Retorna: `void`

Utiliza el conteo de ticks para determinar el tiempo transcurrido resultando en un retardo exacto. Es bloqueante pues se queda en un bucle while hasta que se cuentan los ticks necesarios para lograr el tiempo especificado.

Retardo no bloqueante

Este tipo de retardo permite realizar otras tareas mientras se ejecuta ya que simplemente se chequea si el tiempo de retardo se ha arribado en lugar de quedarse bloqueado esperando a que se complete el tiempo como en los casos anteriores.

Define el tipo de datos estructurado `delay_t`

Contiene las funciones:

```
void delayConfig( delay_t * delay, tick_t duration );
```

- Parámetro: `delay_t * delay` dirección de memoria de una variable del tipo delay_t.
- Parámetro: `tick_t duration` tiempo de duración del retardo en ms.
- Retorna: `void`

```
bool_t delayRead( delay_t * delay );
```

- Parámetro: `delay_t * delay` dirección de memoria de una variable del tipo delay_t.
- Retorna: `bool_t` TRUE cuando el delay se cumplió, FALSE en caso contrario.

```
void delayWrite( delay_t * delay, tick_t duration );
```

- Parámetro: `delay_t * delay` dirección de memoria de una variable del tipo delay_t.
- Parámetro: `tick_t duration` tiempo de duración del retardo en ms.
- Retorna: `void`

Uso:

Se utiliza declarando una variable de estructura del tipo delay_t, por ejemplo:

```
delay_t myDelay;
```

Luego, se configura inicialmente pasando como parámetro la variable recién declarada

```
delayConfig( &myDelay, 500 );
```

Se detecta con un bloque if si se cumplió el delay leyéndolo con

```
delayRead( &myDelay );
```

La primera vez que se ejecuta delayRead activa el mismo. delayRead devuelve TRUE cuando se completo y se vuelve a relanzar automáticamente.

Con `delayWrite( &myDelay, 1000 );` se puede cambiar la duración de un delay en tiempo de ejecución.

sAPI_DigitalIO

Manejo de Entradas y Salidas digitales.

Configuración inicial y modo de una entrada o salida

```
bool_t digitalConfig( int8_t pin, int8_t config);
```


- Parámetro: `int8_t pin` pin a configurar (ver Digital IO Map).

- Parámetro: `int8_t config` configuración.
- Retorna: `bool_t` TRUE si la configuración es correcta.

Posibles configuraciones:

- `ENABLE_DIGITAL_IO` Habilita las entradas y salidas digitales.
- `INPUT, INPUT_PULLUP, INPUT_PULLDOWN, INPUT_REPEATER` Pin configurado como entrada digital en sus distintas variantes.
- `OUTPUT` Pin configurado como salida digital.

Lectura de Entrada digital

```
bool_t digitalRead( int8_t pin );
```

- Parámetro: `int8_t pin` pin a leer (ver Digital IO Map).
- Retorna: `bool_t` valor de la entrada digital.

Escritura de Salida Digital

```
bool_t digitalWrite( int8_t pin, bool_t value );
```

- Parámetro: `int8_t pin` pin a escribir (ver Digital IO Map).
- Parámetro: `bool_t value` valor a escribir en el pin.
- Retorna: `bool_t` FALSE en caso de errores.

sAPI_AnalogIO

Manejo de Entradas y Salidas analógicas.

Configuración inicial de entradas o salidas analógicas

```
void analogConfig( uint8_t config );
```

- Parámetro: `uint8_t config` configuración.
- Retorna: `void`.

Posibles configuraciones:

- `ENEABLE_ANALOG_INPUTS` Habilita las entradas analógicas.
- `DISABLE_ANALOG_INPUTS` Deshabilita las entradas analógicas.
- `ENEABLE_ANALOG_OUTPUTS` Habilita las salidas analógicas.
- `DISABLE_ANALOG_OUTPUTS` Deshabilita las salidas analógicas.

Lectura de Entrada analógica

```
uint16_t analogRead( uint8_t analogInput );
```

- Parámetro: `uint8_t analogInput` pin a leer (ver Analog IO Map).
- Retorna: `uint16_t` el valor actual de la entrada analógica.

Escritura de Salida analógica

```
void analogWrite( uint8_t , uint16_t value );
```

- Parámetro: `uint8_t analogOutput` pin a escribir (ver Analog IO Map).
- Parámetro: `uint16_t value` valor del pin a escribir.
- Retorna: `void`.

sAPI_Uart

Manejo del periférico de comunicación UART (puerto serie asincrónico).

Configuración

```
void uartConfig( uint8_t uart, uint32_t baudRate );
```

- Parámetro: `uint8_t uart` UART a configurar (ver Uart Map).
- Parámetro: `uint32_t baudRate` tasa de bits.
- Retorna: `void`.

Posibles configuraciones de baudRate: `9600, 57600, 115200, etc.`

Recibir Byte

```
uint8_t uartReadByte( uint8_t uart );
```

- Parámetro: `uint8_t uart` UART a configurar (ver Uart Map).
- Retorna: `uint8_t` 0 si no hay dato recibido o el Byte recibido.

Enviar Byte

```
void uartWriteByte( uint8_t uart, uint8_t byte );
```

- Parámetro: `uint8_t uart` UART a configurar (ver Uart Map).
- Parámetro: `uint8_t byte` Byte a enviar.
- Retorna: `void`.

Enviar String

```
void uartWriteString( uint8_t uart, uint8_t * str );
```

- Parámetro: `uint8_t uart` UART a configurar (ver Uart Map).
- Parámetro: `uint8_t * str` String a enviar, puede ser un literal, por ejemplo "hola", o un vector de uint8_t terminado en 0 o '\0' (caracter NULL).
- Retorna: `void`.

sAPI_I2c

Manejo del periférico bus comunicación I2C (Inter Integrated Circuits).

Configuración

```
bool_t i2cConfig( uint8_t i2cNumber, uint32_t clockRateHz );
```

- Parámetro: `uint8_t i2cNumber` ID de periférico I2C a configurar (ver I2C Map). Por ahora funciona únicamente el I2C0.
- Parámetro: `uint32_t clockRateHz` configuración de velocidad del bus I2C.
- Retorna: `bool_t` TRUE si la configuración es correcta.

Posibles configuraciones de clockRateHz: 100000, etc.

Lectura

```
bool_t i2cWrite( uint8_t i2cNumber, uint8_t addr, uint8_t record, uint8_t * buf, uint16_t len );
```

- Parámetro: `uint8_t i2cNumber` ID de periférico I2C a leer (ver I2C Map). Por ahora funciona únicamente el I2C0.
- Parámetro: `uint8_t addr` Dirección del sensor conectado por I2C a leer.
- Parámetro: `uint8_t record` Registro a leer.
- Parámetro: `uint8_t * buf` puntero al buffer donde se almacenarán los datos leídos.
- Parámetro: `uint16_t len` tamaño del buffer donde se almacenarán los datos leídos.
- Retorna: `bool_t` TRUE si se pudo leer correctamente.

Escritura

```
bool_t i2cRead( uint8_t i2cNumber, uint8_t addr, uint8_t record, uint8_t * buf, uint16_t len );
```

- Parámetro: `uint8_t i2cNumber` ID de periférico I2C a escribir (ver I2C Map). Por ahora funciona únicamente el I2C0.
- Parámetro: `uint8_t addr` Dirección del sensor conectado por I2C a escribir.
- Parámetro: `uint8_t record` Registro a escribir.
- Parámetro: `uint8_t * buf` puntero al buffer donde se encuentran los datos a escribir.
- Parámetro: `uint16_t len` tamaño del buffer donde se encuentran los datos a escribir.
- Retorna: `bool_t` TRUE si se pudo escribir correctamente.

sAPI_Rtc

Manejo del periférico RTC (reloj de tiempo real).

Configuración

```
bool_t rtcConfig( RTC_t * rtc );
```

- Parámetro: `RTC_t * rtc` Puntero a estructura de configuración del tipo RTC_t.
- Retorna: `bool_t` TRUE si la configuración es correcta.

La estructura del tipo `RTC_t` contiene los parámetros:

- `uint16_t year` año, con valores desde 1 a 4095.
- `uint8_t month` mes, con valores desde 1 a 12.
- `uint8_t mday` día, con valores desde 1 a 31.
- `uint8_t wday` día de la semana, con valores desde 1 a 7.
- `uint8_t hour` horas, con valores desde 0 a 23.
- `uint8_t min` minutos, con valores desde 0 a 59.
- `uint8_t sec` segundos, con valores desde 0 a 59.

Lectura de fecha y hora

```
bool_t rtcRead( RTC_t * rtc );
```

- Parámetro: `RTC_t * rtc` Puntero a estructura del tipo RTC_t donde se guarda la fecha y hora.
- Retorna: `bool_t` TRUE.

Establecer la fecha y hora

```
bool_t rtcWrite( RTC_t * rtc );
```

- Parámetro: `RTC_t * rtc` Puntero a estructura del tipo RTC_t con la nueva fecha y hora a setear.
- Retorna: `bool_t` TRUE.

sAPI_Pwm

Manejo de salidas PWM (modulación por ancho de pulso). En la EDU-CIAA-NXP se utiliza internamente el periférico SCT para generar los PWM.

Configuración

```
bool_t pwmConfig( uint8_t pwmNumber, uint8_t config);
```

- Parámetro: `uint8_t pwmNumber` pin a configurar como salida PWM (ver PWM Map).
- Parámetro: `uint8_t config` configuración.
- Retorna: `bool_t` TRUE si la configuración es correcta.

Posibles configuraciones:

- `ENABLE_PWM_TIMERS` habilita el o los Timers en modo PWM.
- `DISABLE_PWM_TIMERS` deshabilita el o los Timers en modo PWM.
- `ENABLE_PWM_OUTPUT` habilita la salida PWM particular.
- `DISABLE_PWM_OUTPUT` deshabilita la salida PWM particular.

Lectura del ciclo de trabajo (duty cycle) de la salida PWM

```
uint8_t pwmRead( uint8_t pwmNumber );
```

- Parámetro: `uint8_t pwmNumber` salida PWM a leer el ciclo de trabajo.
- Retorna: `uint8_t` el ciclo de trabajo de la salida PWM.

Establecer el ciclo de trabajo de la salida PWM

```
bool_t pwmWrite( uint8_t pwmNumber, uint8_t percent );
```

- Parámetro: `uint8_t pwmNumber` salida PWM a leer el ciclo de trabajo.
- Parámetro: `uint8_t percent` valor de ciclo de trabajo a setear en la salida PWM.
- Retorna: `bool_t` TRUE.

sAPI_Servo

Manejo de salidas para Servomortores angulares (usan modulación por ancho de pulso). En la EDU-CIAA-NXP se utilizan internamente los periféricos TIMER para generar estas salidas.

Configuración

```
bool_t servoConfig( uint8_t servoNumber, uint8_t config );
```

- Parámetro: `uint8_t servoNumber` pin a configurar como salida Servo (ver Servo Map).
- Parámetro: `uint8_t config` configuración.
- Retorna: `bool_t` TRUE si la configuración es correcta.

Posibles configuraciones:

- `ENABLE_SERVO_TIMERS` habilita el o los Timers en modo PWM para Servo.
- `DISABLE_SERVO_TIMERS` deshabilita el o los Timers en modo PWM para Servo.
- `ENABLE_SERVO_OUTPUT` habilita la salida PWM particular.
- `DISABLE_SERVO_OUTPUT` deshabilita la salida PWM particular.

Lectura del valor angular actual de la salida Servo

```
uint8_t servoRead( uint8_t servoNumber);
```

- Parámetro: `uint8_t servoNumber` pin como salida Servo a leer.
- Retorna: `bool_t` el valor angular actual de la salida Servo (de 0 a 180°).

Establecer el valor angular de la salida Servo

```
bool_t servoWrite( uint8_t servoNumber, uint8_t angle );
```

- Parámetro: `uint8_t servoNumber` pin como salida Servo a escribir.
- Parámetro: `uint8_t angle` valor angular a establecer en la salida Servo (de 0 a 180°).
- Retorna: `bool_t` TRUE.

sAPI_Hmc5883l

Manejo del sensor magnetómetro vectorial (x,y,z) HMC5883L de Honeywell. Este sensor se conecta mediante I2C.

Configuración

```
bool_t hmc5883lPrepareDefaultConfig( HMC5883L_config_t * config );
```

- Parámetro: `HMC5883L_config_t *config` puntero a estructura del tipo HMC5883L_config_t a donde se cargarán los valores por defecto de configuración.
- Retorna: `bool_t` TRUE.

```
bool_t hmc5883lConfig( HMC5883L_config_t config );
```

- Parámetro: `HMC5883L_config_t *config` estructura del tipo HMC5883L_config_t desde donde se cargarán los valores de configuración.
- Retorna: `bool_t` TRUE si la configuración es correcta.

La estructura del tipo `HMC5883L_config_t` contiene:

- `HMC5883L_samples_t samples` Numero de muestras que promedia para calcular la salida de la medición. Valores admitidos:
  - HMC5883L_1_sample
  - HMC5883L_2_sample
  - HMC5883L_4_sample
  - HMC5883L_8_sample
  - HMC5883L_DEFAULT_sample = HMC5883L_1_sample
- `HMC5883L_rate_t rate` Bits de tasa de datos de salida. Estos bits establecen la tasa de escritura de los 3 registros de datos de salida del sensor. Valores admitidos:
  - HMC5883L_0_75_Hz
  - HMC5883L_1_50_Hz
  - HMC5883L_3_Hz
  - HMC5883L_7_50_Hz
  - HMC5883L_15_Hz

    - HMC5883L_30_Hz
    - HMC5883L_75_Hz
    - HMC5883L_DEFAULT_rate = HMC5883L_15_Hz
- `HMC5883L_messurement_t meassurement` Bits de configuración de medición. Estos bits definen el flujo de medición del sensor. Específicamente si se aplica, o no, un bias a la medición. Valores admitidos:
    - HMC5883L_normal
    - HMC5883L_positive
    - HMC5883L_regative
    - HMC5883L_DEFAULT_messurement = HMC5883L_normal
- `HMC5883L_gain_t gain` Bits de configuración de ganancia. Estos bits configuran la ganancia del sensor. Esta configuración se aplica a todos los canales. Valores admitidos:
    - HMC5883L_1370 para ± 0.88 Ga
    - HMC5883L_1090 para ± 1.3 Ga
    - HMC5883L_820 para ± 1.9 Ga
    - HMC5883L_660 para ± 2.5 Ga
    - HMC5883L_440 para ± 4.0 Ga
    - HMC5883L_390 para ± 4.7 Ga
    - HMC5883L_330 para ± 5.6 Ga
    - HMC5883L_230 para ± 8.1 Ga
    - HMC5883L_DEFAULT_gain = HMC5883L_1090
- `HMC5883L_mode_t mode`. Modo de medición. Valores admitidos:
    - HMC5883L_continuous_measurement
    - HMC5883L_single_measurement
    - HMC5883L_idle
    - HMC5883L_DEFAULT_mode = HMC5883L_single_measurement

```
bool_t hmc5883lIsAlive( void );
```

- Parámetro: `void` ninguno.
- Retorna: `bool_t` TRUE si puede comunicarse con el sensor.

```
bool_t hmc5883lRead( int16_t * x, int16_t * y, int16_t * z );
```

- Parámetro: `int16_t * x` puntero entero de 16 bits con signo donde se guardará el valor leído del sensor HMC5883L en la componente x.
- Parámetro: `int16_t * y` puntero entero de 16 bits con signo donde se guardará el valor leído del sensor HMC5883L en la componente y.
- Parámetro: `int16_t * z` puntero entero de 16 bits con signo donde se guardará el valor leído del sensor HMC5883L en la componente z.
- Retorna: `bool_t` TRUE si puede leer correctamente el sensor magnetómetro.

Archivos que componen la biblioteca

src (.c):

- sAPI_AnalogIO.c

- sAPI_Board.c
- sAPI_DataTypes.c
- sAPI_Delay.c
- sAPI_DigitalIO.c
- sAPI_Hmc5883l.c
- sAPI_I2c.c
- sAPI_IsrVector.c
- sAPI_Pwm.c
- sAPI_Rtc.c
- sAPI_Sct.c
- sAPI_Servo.c
- sAPI_Spi.c
- sAPI_Tick.c
- sAPI_Timer.c
- sAPI_Uart.c

inc (.h):

- sAPI_AnalogIO.h
- sAPI_Board.h
- sAPI_DataTypes.h
- sAPI_Delay.h
- sAPI_DigitalIO.h
- sAPI_Hmc5883l.h
- sAPI_I2c.h
- sAPI_IsrVector.h
- sAPI_PeripheralMap.h
- sAPI_Pwm.h
- sAPI_Rtc.h
- sAPI_Sct.h
- sAPI_Servo.h
- sAPI_Spi.h
- sAPI_Tick.h
- sAPI_Timer.h
- sAPI_Uart.h
- sAPI.h