API de la biblioteca sAPI

Módulos

sAPI_Config

Contiene configuraciones de la biblioteca.

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`

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 );
```

- Parámetros: `void`
- Retorna: `void`

sAPI_PeripheralMap

Contiene el mapa de periféricos.

DigitalIO Map

```
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
```

AnalogIO Map

```
AI0, AI1, AI2, AO
```

Uart Map

```
UART_USB, UART_232, UART_485
```

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);
```

- Parámetros: `int8_t pin, int8_t config`
- Retorna: `bool_t` TRUE si la configuración es correcta.

Configuraciones:

```
INITIALIZE
```

```
INPUT, INPUT_PULLUP, INPUT_PULLDOWN, INPUT_REPEATER
```

```
OUTPUT
```

Lectura de Entrada digital

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

- Parámetros: `int8_t pin`
- Retorna: `bool_t` valor de la entrada digital.

Escritura de Salida Digital

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

- Parámetros: `int8_t pin, bool_t value`
- Retorna: `bool_t` FALSE en caso de errores.

sAPI_Tick

Configuración de interrupción periódica

```
bool_t tickConfig( tick_t tickRateMSvalue );
```

- Parámetros: `tick_t tickRateMSvalue`
- Retorna: `bool_t` 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 funcin 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.

Leer la variable del conteo actual de ticks

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

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

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 );
```

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

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

Función que se ejecuta en cada vez que ocurre un tick

```
void tickHook( void );
```

- Parámetros: `void`
- Retorna: `void`

Esta función debe declararla el usuario en su programa. Para poder utilizarla, se debe configurar en **sAPI_Config.h**

```
#define SAPI_USE_TICK_HOOK TRUE
```

de esta manera se le da aviso a la biblioteca que se encargará de ejecutarla automáticamente a cada tick.

sAPI_Delay

Para utilizar los retardos (con excepción del retardo inexacto) se debe configurar el Tick.

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`
- 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 );
```

- Parámetros: `delay_t * delay, tick_t duration`
- Retorna: `void`

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

- Parámetros: `delay_t * delay`
- Retorna: `bool_t` TRUE cuando el delay se cumplió, FALSE en caso contrario.

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

- Parámetros: `delay_t * delay, tick_t duration`
- 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.

Archivos que componen la biblioteca

src (.c):

- sAPI_AnalogIO.c
- sAPI_Board.c
- sAPI_Delay.c
- sAPI_DigitalIO.c
- sAPI_IsrVector.c
- sAPI_Tick.c
- sAPI_Uart.c

inc (.h):

- sAPI.h
- sAPI_AnalogIO.h
- sAPI_Board.h
- sAPI_Config.h
- sAPI_DataTypes.h
- sAPI_Delay.h
- sAPI_DigitalIO.h
- sAPI_IsrVector.h
- sAPI_PeripheralMap.h
- sAPI_Tick.h
- sAPI_Uart.h