



# **HAL Drivers Application Programming Interface**

Document Number: F8W-2005-1504

**Texas Instruments, Inc.**  
San Diego, California USA  
(619) 542-1200

Version	Description	Date
1.0	Initial release.	03/02/2006
1.1	Hal sleep updates	11/20/2006
1.2	Updated LCD and timers API	11/22/2006

## TABLE OF CONTENTS

<b>1. INTRODUCTION .....</b>	<b>1</b>
1.1 PURPOSE.....	1
1.2 ACRONYMS.....	1
<b>2. DRIVERS API OVERVIEW .....</b>	<b>2</b>
2.1 FUNCTION CALLS.....	2
2.1.1 Initialization Function Calls.....	2
2.1.2 Service Access Function Calls.....	2
2.1.3 Callback Function Calls.....	2
2.2 SERVICES.....	2
<b>3. ADC SERVICE.....</b>	<b>3</b>
3.1 HALADCINIT () .....	3
3.1.1 Description .....	3
3.1.2 Prototype .....	3
3.1.3 Parameter Details.....	3
3.1.4 Return .....	3
3.2 HALADCREAD () .....	3
3.2.1 Description .....	3
3.2.2 Prototype .....	3
3.2.3 Parameter Details.....	3
3.2.4 Return .....	3
3.3 CONSTANTS .....	3
3.3.1 Channels .....	3
3.3.2 Resolutions.....	4
<b>4. LCD SERVICE .....</b>	<b>4</b>
4.1 HALLCDINIT() .....	4
4.1.1 Description .....	4
4.1.2 Prototype .....	4
4.1.3 Parameter Details.....	4
4.1.4 Return .....	4
4.2 HALLCDWRITESTRING() .....	4
4.2.1 Description .....	4
4.2.2 Prototype .....	4
4.2.3 Parameter Details.....	4
4.2.4 Return .....	5
4.3 HALLCDWRITEVALUE().....	5
4.3.1 Description .....	5
4.3.2 Prototype .....	5
4.3.3 Parameter Details.....	5
4.4 HALLCDWRITESCREEN() .....	5
4.4.1 Description .....	5
4.4.2 Prototype .....	5
4.4.3 Parameter Details.....	5
4.4.4 Return .....	5
4.5 HALLCDWRITESTRINGVALUE() .....	5
4.5.1 Description .....	5
4.5.2 Prototype .....	5
4.5.3 Parameter Details.....	6
4.5.4 Return .....	6

4.6	HALLCDWRITESTRINGVALUEVALUE()	6
4.6.1	Description	6
4.6.2	Prototype	6
4.6.3	Parameter Details	6
4.6.4	Return	6
4.7	HALLCDDISPLAYPERCENTBAR()	7
4.7.1	Description	7
4.7.2	Prototype	7
4.7.3	Parameter Details	7
4.7.4	Return	7
4.8	CONSTANTS	7
4.8.1	Options	7
<b>5.</b>	<b>LED SERVICE</b>	<b>7</b>
5.1	HALLEDINIT()	7
5.1.1	Description	7
5.1.2	Prototype	7
5.1.3	Parameter Details	7
5.1.4	Return	8
5.2	HALLEDSET()	8
5.2.1	Description	8
5.2.2	Prototype	8
5.2.3	Parameter Details	8
5.2.4	Return	8
5.3	HALLEDBLINK()	8
5.3.1	Description	8
5.3.2	Prototype	8
5.3.3	Parameter Details	8
5.3.4	Return	8
5.4	HALLEDGETSTATE()	8
5.4.1	Description	8
5.4.2	Prototype	9
5.4.3	Parameter Details	9
5.4.4	Return	9
5.5	HALLEDENTERSLEEP()	9
5.5.1	Description	9
5.5.2	Prototype	9
5.5.3	Parameter Details	9
5.5.4	Return	9
5.6	HALLEDEXIT_SLEEP()	9
5.6.1	Description	9
5.6.2	Prototype	9
5.6.3	Parameter Details	9
5.6.4	Return	9
5.7	CONSTANTS	9
5.7.1	LEDs	9
5.7.2	Modes	10
<b>6.</b>	<b>KEY SERVICE</b>	<b>10</b>
6.1	HALKEYINIT()	10
6.1.1	Description	10
6.1.2	Prototype	10
6.1.3	Parameter Details	10
6.1.4	Return	10
6.2	HALKEYCONFIG()	10

6.2.1	Description .....	10
6.2.2	Prototype .....	11
6.2.3	Parameter Details.....	11
6.2.4	Return .....	11
6.3	HALKEYREAD() .....	11
6.3.1	Description .....	11
6.3.2	Prototype .....	11
6.3.3	Parameter Details.....	11
6.3.4	Return .....	11
6.4	HALKEYENTERSLEEP() .....	11
6.4.1	Description .....	11
6.4.2	Prototype .....	11
6.4.3	Parameter Details.....	11
6.4.4	Return .....	11
6.5	HALKEYEXITSLLEEP() .....	12
6.5.1	Description .....	12
6.5.2	Prototype .....	12
6.5.3	Parameter Details.....	12
6.5.4	Return .....	12
6.6	HALKEYPOLL() .....	12
6.6.1	Description .....	12
6.6.2	Prototype .....	12
6.6.3	Parameter Details.....	12
6.6.4	Return .....	12
6.7	HALKEYPRESSED() .....	12
6.7.1	Description .....	12
6.7.2	Prototype .....	12
6.7.3	Parameter Details.....	12
6.7.4	Return .....	12
6.8	CONSTANT.....	12
6.8.1	Keys .....	12
6.8.2	Joystick .....	13
6.8.3	States.....	13
<b>7.</b>	<b>SLEEP SERVICE.....</b>	<b>13</b>
7.1	HALSLEEP().....	13
7.1.1	Description .....	13
7.1.2	Prototype .....	13
7.1.3	Parameter Details.....	13
7.1.4	Return .....	13
7.2	HALSLEEPWAIT().....	14
7.2.1	Description .....	14
7.2.2	Prototype .....	14
7.2.3	Parameter Details.....	14
7.2.4	Return .....	14
<b>8.</b>	<b>TIMER SERVICE .....</b>	<b>14</b>
8.1	OPERATION MODES .....	14
8.2	CHANNELS.....	14
8.3	INTERRUPTS/CHANNEL MODES .....	14
8.4	HALTIMERINIT ().....	14
8.4.1	Description .....	14
8.4.2	Prototype .....	14
8.4.3	Parameter Details.....	14
8.4.4	Return .....	14

8.5	HALTIMERCONFIG()	15
8.5.1	Description	15
8.5.2	Prototype	15
8.5.3	Parameter Details	15
8.5.4	Return	15
8.6	HALTIMERSTART()	15
8.6.1	Description	15
8.6.2	Prototype	15
8.6.3	Parameter Details	16
8.6.4	Return	16
8.7	HALTIMERSTOP()	16
8.7.1	Description	16
8.7.2	Prototype	16
8.7.3	Parameter Details	16
8.7.4	Return	16
8.8	HALTIMERTICK()	16
8.8.1	Description	16
8.8.2	Prototype	16
8.8.3	Parameter Details	16
8.8.4	Return	16
8.9	HALTIMERINTERRUPTENABLE()	16
8.9.1	Description	16
8.9.2	Prototype	16
8.9.3	Parameter Details	17
8.9.4	Return	17
8.10	CONSTANTS	17
8.10.1	Timer ID	17
8.10.2	Channels	17
8.10.3	Channel Modes	17
8.10.4	Operation Modes	18
8.10.5	Prescale	18
8.10.6	Status	18
<b>9.</b>	<b>UART SERVICE</b>	<b>19</b>
9.1	HALUARTINIT()	19
9.1.1	Description	19
9.1.2	Prototype	19
9.1.3	Parameter Details	19
9.1.4	Return	19
9.2	HALUARTOPEN()	19
9.2.1	Description	19
9.2.2	Prototype	19
9.2.3	Parameter Details	19
9.2.4	Return	21
9.3	HALUARTCLOSE()	21
9.3.1	Description	21
9.3.2	Prototype	21
9.3.3	Parameter Details	21
9.3.4	Return	21
9.4	HALUARTREAD()	21
9.4.1	Description	21
9.4.2	Prototype	21
9.4.3	Parameter Details	21
9.4.4	Return	22
9.5	HALUARTWRITE()	22

9.5.1	Description .....	22
9.5.2	Prototype .....	22
9.5.3	Parameter Details.....	22
9.5.4	Return .....	22
9.6	HALUARTIOCTL () .....	22
9.6.1	Description .....	22
9.6.2	Prototype .....	22
9.6.3	Parameter Details.....	22
9.6.4	Return .....	23
9.7	HALUARTPOLL() .....	23
9.7.1	Description .....	23
9.7.2	Prototype .....	23
9.7.3	Parameter Details.....	23
9.7.4	Return .....	23
9.8	HAL_UART_RXBUFFLEN().....	23
9.8.1	Description .....	23
9.8.2	Prototype .....	23
9.8.3	Parameter Details.....	24
9.8.4	Return .....	24
9.9	HAL_UART_TXBUFFLEN() .....	24
9.9.1	Description .....	24
9.9.2	Prototype .....	24
9.9.3	Parameter Details.....	24
9.9.4	Return .....	24
9.10	HAL_UART_FLOWCONTROLSET() .....	24
9.10.1	Description .....	24
9.10.2	Prototype .....	24
9.10.3	Parameter Details.....	24
9.10.4	Return .....	24
9.11	CONSTANTS .....	24
9.11.1	UART Ports.....	24
9.11.2	Baud Rates.....	25
9.11.3	Parity .....	25
9.11.4	Stop Bits.....	25
9.11.5	Status .....	25
9.11.6	Callback Events .....	25

## 1. Introduction

### 1.1 Purpose

This document describes the application programming interface for HAL Drivers. The API provides application the interface to access timers, GPIO, UART, and ADC. This is a platform independent API that provides a superset of features for each service. Not all features will be available for all platforms.

### 1.2 Acronyms

ADC	Analog to Digital Conversion
API	Application Programming Interface.
CD	Carrier Detect
CTS	Clear To Send
DSR	Data Set Ready
DTR	Data Terminal Ready
GPIO	General Purpose Input Output
HAL	Hardware Abstract Layer
MAC	Medium Access Control.
OSAL	Operating System Abstraction Layer.
RI	Ring Indicator
RTS	Ready To Send



## 2. Drivers API Overview

### 2.1 Function Calls

#### 2.1.1 Initialization Function Calls

These function calls are used to initialize a service and /or to setup optional parameters for platform-specific data. Initialization functions are often called at the beginning stage when the device power up.

#### 2.1.2 Service Access Function Calls

These function calls can directly access hardware registers to get/set certain value of the hardware (i.e. ADC) or control the hardware components (i.e. LED).

#### 2.1.3 Callback Function Calls

These functions must be implemented by the application and are used to pass events that generated by the hardware (interrupts, counters, timers...) or by polling mechanism (UART poll, Timer poll...) to upper layer. Data accessed through callback function parameters (such as a pointer to data) are only valid for the execution of the function and should not be considered valid when the function returns. If these functions execute in the context of the interrupt, it must be efficient and not perform CPU-intensive operations or use critical sections.

### 2.2 Services

HAL drivers provide Timer, GPIO, LEDs, Switched, UART, and ADC service for MAC and upper layers. Not all the features in the service are available in every platform. Features in each service can be configured for different platform through initialization function.

### 3. ADC Service

This service supports 8, 10, and 14-bit analog to digital conversion on 8 channels (0-7)

#### 3.1 HalAdcInit ()

##### 3.1.1 Description

This ADC initialization function is called once at the startup. This function has to be called before any other ADC functions can be called. It enables ADC to be initialized with both required and optional parameters.

##### 3.1.2 Prototype

```
void HalAdcInit (void)
```

##### 3.1.3 Parameter Details

None.

##### 3.1.4 Return

None.

#### 3.2 HalAdcRead ()

##### 3.2.1 Description

This function reads and returns the value of the ADC conversion at specified channel and resolution.

##### 3.2.2 Prototype

```
uint16 HalAdcRead (uint8 channel,  
                  uint8 resolution);
```

##### 3.2.3 Parameter Details

**channel** – input channels ([Check Channels Table](#))

**resolution** – resolution of the conversion. ([Check Resolutions Table](#))

##### 3.2.4 Return

Return 16-bit value of the conversion at the given channel and resolution.

### 3.3 Constants

#### 3.3.1 Channels

Parameter	Description
HAL_ADC_CHANNEL_0	Input channel 0
HAL_ADC_CHANNEL_1	Input channel 1
HAL_ADC_CHANNEL_2	Input channel 2
HAL_ADC_CHANNEL_3	Input channel 3

HAL_ADC_CHANNEL_4	Input channel 4
HAL_ADC_CHANNEL_5	Input channel 5
HAL_ADC_CHANNEL_6	Input channel 6
HAL_ADC_CHANNEL_7	Input channel 7

### 3.3.2 Resolutions

Parameter	Description
HAL_ADC_RESOLUTION_8	8-bit resolution
HAL_ADC_RESOLUTION_10	10-bit resolution
HAL_ADC_RESOLUTION_12	12-bit resolution
HAL_ADC_RESOLUTION_14	14-bit resolution

## 4. LCD Service

This service allows writing text on the LCD. Not every board supports LCD.

### 4.1 HalLcdInit()

#### 4.1.1 Description

This initialization function is called once at the startup. This function has to be called before any other LCD function can be called. It enables LCD to be initialized with both required and optional parameters.

#### 4.1.2 Prototype

```
void HalLcdInit (void);
```

#### 4.1.3 Parameter Details

None.

#### 4.1.4 Return

None.

### 4.2 HalLcdWriteString()

#### 4.2.1 Description

This routine writes a string to the LCD.

#### 4.2.2 Prototype

```
void HalLcdWriteString (uint8* str,  
                        uint8 option);
```

#### 4.2.3 Parameter Details

**str** – pointer to the string that will be displayed on the LCD. Max length of the string is defined under HAL\_LCD\_MAX\_CHARS. If the length is greater than HAL\_LCD\_MAX\_CHARS, then only HAL\_LCD\_MAX\_CHARS will be displayed.

**option** – option for the string to be displayed on the LCD. ([Check Options table](#)).

#### 4.2.4 Return

None.

### 4.3 HalLcdWriteValue()

#### 4.3.1 Description

This routine writes a 32-bit value to the LCD.

#### 4.3.2 Prototype

```
void HalLcdWriteValue (uint32 value,  
                      uint8 radix,  
                      uint8 option);
```

#### 4.3.3 Parameter Details

**value** – value that will be displayed on the LCD.

**radix** – representation of the value. It can be 8, 10 or 16. (Octal, Decimal, or Hex)

**option** – option for the value to be displayed on the LCD. ([Check Options table](#))

### 4.4 HalLcdWriteScreen()

#### 4.4.1 Description

This routine writes 2 lines of text on the LCD display.

#### 4.4.2 Prototype

```
void HalLcdWriteScreen( char *line1,  
                      char *line2 );
```

#### 4.4.3 Parameter Details

Line1 – pointer to the 1<sup>st</sup> string that will be displayed on the LCD

Line2 – pointer to the 2<sup>nd</sup> string that will be displayed on the LCD

#### 4.4.4 Return

None.

### 4.5 HalLcdWriteStringValue()

#### 4.5.1 Description

This routine writes a string followed by a 16-bit value on the specified line number on the LCD display.

#### 4.5.2 Prototype

```
void HalLcdWriteStringValue( char *title,
```

```
uint16 value,
uint8 format,
uint8 line );
```

#### 4.5.3 Parameter Details

**title** – string that will be displayed on the LCD display.

**value** – value that will be displayed following the “title”.

**format** – format of the value that will be displayed. It can be 8, 10, and 16 (Octal, Decimal and Hex)

**line** – the line number where the string (title and value) will be displayed on the LCD display.

Example: To display “Count: 180” on line 2 of the LCD display

```
HalLcdWriteStringValue (“Count:”, 180, 10, 2);
```

#### 4.5.4 Return

None.

### 4.6 HalLcdWriteStringValueValue()

#### 4.6.1 Description

Write two 16-bit values back to back on the specified line on the LCD display underneath the a title.

#### 4.6.2 Prototype

```
void HalLcdWriteStringValueValue( char *title,
uint16 value1,
uint8 format1,
uint16 value2,
uint8 format2,
uint8 line );
```

#### 4.6.3 Parameter Details

**title** – string that will be displayed on the LCD display.

**value1** – 1<sup>st</sup> value that will be displayed after the “title”.

**format1** – format for the “value1”. It can be 8, 10, and 16 (Octal, Decimal, and Hex).

**value2** – 2<sup>nd</sup> value that will be displayed after “value1”.

**format2** – format for the “value2”. It can be 8, 10, and 16 (Octal, Decimal, and Hex).

**line** – line number where the string (title, value1 and value2) will be displayed on the LCD display.

Example: HalLcdWriteStringValueValue (“Test: ”, 2, 10, 30, 10, 1);

Test: 2 30

#### 4.6.4 Return

None.

## 4.7 HalLcdDisplayPercentBar()

### 4.7.1 Description

Simulating a percentage bar on the LCD with the percentage in numerical figure in the middle of the bar.

### 4.7.2 Prototype

```
void HalLcdDisplayPercentBar( char *title,
                             uint8 value );
```

### 4.7.3 Parameter Details

**title** – string that will be displayed on the 1<sup>st</sup> line of the LCD display

**value** – percentage value that will be displayed in the middle of the bar

Example: `HalLcdDisplayPercentBar ("Rate:", 50);`

Display:   Rate:  
          [|||||||50+          ]

### 4.7.4 Return

None.

## 4.8 Constants

### 4.8.1 Options

Option	Description
HAL_LCD_LINE_1	Display the text on line 1 of the LCD
HAL_LCD_LINE_2	Display the text on line 2 of the LCD

## 5. LED Service

This service allows LEDs to be controlled in different ways. LED service supports ON, OFF, TOGGLE, FLASH and BLINK. Not all platforms support these modes.

### 5.1 HalLedInit()

#### 5.1.1 Description

This LED initialization function is called once at the startup. This function has to be called before any other LED function can be called. It enables LED to be initialized with both required and optional parameters.

#### 5.1.2 Prototype

```
void HalLEDInit (void);
```

#### 5.1.3 Parameter Details

None.

#### 5.1.4 Return

None.

### 5.2 HalLedSet()

#### 5.2.1 Description

This function will set the given LEDs ON, OFF, BLINK, FLASH, or TOGGLE. If BLINK and FLASH mode are used, a set of default parameters will be used. To customize these parameters, HalLedBlink() has to be used.

#### 5.2.2 Prototype

```
void HalLedSet (uint8 led, uint8 mode);
```

#### 5.2.3 Parameter Details

**led** – bit mask of leds to be turned ON ([See LEDs Table](#))

**mode** – new mode for the leds. ([See Modes Table](#))

#### 5.2.4 Return

None.

### 5.3 HalLedBlink()

#### 5.3.1 Description

This function will blink the given LEDs based on the provided parameters.

#### 5.3.2 Prototype

```
void HalLedBlink (uint8 leds,  
                  uint8 numBlinks,  
                  uint8 percent,  
                  uint16 period);
```

#### 5.3.3 Parameter Details

**leds** – bit mask of leds to be blinked ([See LEDs Table](#))

**numBlinks** – number of times the LED will blink

**percent** – percentage of the cycle that is ON.

**period** – time in milliseconds of one ON/OFF cycle.

#### 5.3.4 Return

None.

### 5.4 HalLedGetState()

#### 5.4.1 Description

This function returns the current state of the leds.

## 5.4.2 Prototype

```
uint8 HalLedGetState (void);
```

## 5.4.3 Parameter Details

None.

## 5.4.4 Return

8-bit contains the current state of the leds. Each bit indicates the corresponding led status.

## 5.5 HalLedEnterSleep()

### 5.5.1 Description

This function stores the current state of the leds and turn off all the leds to conserve power. It also sets a global state variable indicating sleep mode has been entered. This global state variable will stop the interrupt from processing the leds during while in sleep mode.

### 5.5.2 Prototype

```
void HalLedEnterSleep (void);
```

### 5.5.3 Parameter Details

None.

### 5.5.4 Return

None.

## 5.6 HalLedExitSleep()

### 5.6.1 Description

This function restores the original state of the leds as before it entering sleep mode

### 5.6.2 Prototype

```
void HalLedExitSleep (void);
```

### 5.6.3 Parameter Details

None.

### 5.6.4 Return

None.

## 5.7 Constants

### 5.7.1 LEDs

LED	Description
HAL_LED_1	LED 1
HAL_LED_2	LED 2



HAL_LED_3	LED 3
HAL_LED_4	LED 4
HAL_LED_ALL	All LEDs

## 5.7.2 Modes

Mode	Description
HAL_LED_MODE_OFF	Turn OFF the LED
HAL_LED_MODE_ON	Turn ON the LED
HAL_LED_MODE_BLINK	BLINK the LED
HAL_LED_MODE_FLASH	FLASH the LED
HAL_LED_MODE_TOGGLE	TOGGLE the LED

## 6. KEY Service

This service provides services for keys, switches and joysticks. The service can be polling or interrupt driven. A callback function has to be registered in order for the service to inform the application the status of the keys/switches/joysticks.

### 6.1 HalKeyInit()

#### 6.1.1 Description

This initialization function is called once at the startup. This function has to be called before any other function that use keys/switches/joysticks can be called. It enables Keys/switches/joysticks to be initialized with both required and optional parameters.

#### 6.1.2 Prototype

```
void HalKeyInit (void *init);
```

#### 6.1.3 Parameter Details

None.

#### 6.1.4 Return

None.

### 6.2 HalKeyConfig()

#### 6.2.1 Description

This function is used to configure the Keys/Switches/Joysticks service as polling or interrupt driven. It also setup a callback function for the service. If interrupt is not used, polling starts automatically after 100ms. Keys/switches/joystick will be polled every 100ms. If interrupt is used, an ISR will handle the case. There is a delay of 25ms after the interrupt occurs to eliminate de-bounce.

### 6.2.2 Prototype

```
void HalKeyConfig (bool interruptEnable,  
                  halKeyCBack_t *cback);
```

### 6.2.3 Parameter Details

**interruptEnable** – TRUE or FALSE. Enable or disable the interrupt. If interrupt is disabled, the keys/switches/joysticks will be polled. Otherwise, the keys/switches/joysticks will be on interrupt.

**cback** – this call back occurs when a key, switch or joystick is active. If the callback is set to NULL, the event will not be handled.

```
typedef void (halGpioCback_t) (uint8 key, uint8 state);
```

**key** – the key/switch/joystick that will cause the callback when it's triggered. ([Check Keys table](#))

**state** – current state of the key/switch/joystick that cause the callback. ([Check States table](#))

### 6.2.4 Return

None.

## 6.3 HalKeyRead()

### 6.3.1 Description

This function is used to read the current state of the Key/Switch/Joystick. If the Key Service is set to polling, this function will be called by the Hal Driver Task every 100ms. If the Key Service is set to interrupt driven, this function will be called by the Hal Driver Task 25ms after the interrupt occurs. If a callback is registered during HalKeyConfig(), that callback will be sent back to the application with the new status of the keys. Otherwise, there will be no further action.

### 6.3.2 Prototype

```
uint8 HalKeyRead ( void );
```

### 6.3.3 Parameter Details

None.

### 6.3.4 Return

Return the bit mask state of the keys. ([Check Keys table](#))

## 6.4 HalKeyEnterSleep()

### 6.4.1 Description

This function sets a global state variable indicating sleep mode has been entered. This global state variable is used to stop the interrupt from processing the keys during sleep mode.

### 6.4.2 Prototype

```
void HalKeyEnterSleep (void);
```

### 6.4.3 Parameter Details

None.

### 6.4.4 Return

None.

## 6.5 HalKeyExitSleep()

### 6.5.1 Description

This function set a global state variable indicating sleep mode has been exit. It also processes those keys that are stored by the key interrupt.

### 6.5.2 Prototype

```
void HalKeyExitSleep (void);
```

### 6.5.3 Parameter Details

None.

### 6.5.4 Return

None.

## 6.6 HalKeyPoll()

### 6.6.1 Description

This routine is used internally by hal driver

### 6.6.2 Prototype

```
void HalKeyPoll ( void );
```

### 6.6.3 Parameter Details

None.

### 6.6.4 Return

None.

## 6.7 HalKeyPressed()

### 6.7.1 Description

This routine is used internally by hal sleep

### 6.7.2 Prototype

```
bool HalKeyPressed( void );
```

### 6.7.3 Parameter Details

None.

### 6.7.4 Return

None.

## 6.8 Constant

### 6.8.1 Keys

Key	Description
HAL_KEY_SW_1	Key #1 is pressed
HAL_KEY_SW_2	Key #2 is pressed
HAL_KEY_SW_3	Key #3 is pressed
HAL_KEY_SW_4	Key #4 is pressed
HAL_KEY_SW_5	Key #5 is pressed
HAL_KEY_SW_6	Key #6 is pressed

### 6.8.2 Joystick

Key	Description
HAL_KEY_UP	Joystick is up
HAL_KEY_RIGHT	Joystick is down
HAL_KEY_CENTER	Joystick is center
HAL_KEY_LEFT	Joystick is left
HAL_KEY_DOWN	Joystick is down

### 6.8.3 States

State	Description
HAL_KEY_STATE_NORMAL	The key is press normally
HAL_KEY_STATE_SHIFT	The key is shift and pressed

## 7. Sleep Service

This service is part of the power saving mechanism. Osal uses these routines to exercise low power mode when POWER\_SAVING symbol is compiled.

### 7.1 HalSleep()

#### 7.1.1 Description

This routine is called from the OSAL task loop through the OSAL interface to set the low power mode of the MAC.

#### 7.1.2 Prototype

```
void halSleep(uint16 osal_timeout)
```

#### 7.1.3 Parameter Details

osal\_timeout – The next OSAL timer timeout. This will be used to determine how long the MAC needs to sleep or awake.

#### 7.1.4 Return

None.

## 7.2 HalSleepWait()

### 7.2.1 Description

This routine performs a blocking wait.

### 7.2.2 Prototype

```
void halSleepWait(uint16 duration)
```

### 7.2.3 Parameter Details

duration – duration of the blocking in micro seconds

### 7.2.4 Return

None.

## 8. Timer Service

This service supports up to four hardware timers, two 8-bit timers and two 16-bit timers.

### 8.1 Operation Modes

Timer service supports Normal and CTC (Clear Timer on Compare) mode. In Normal mode, the Timer/Counter is always up, and no Timer/Counter clear is performed. The Timer/Counter simply overruns when it passes its maximum and then restarts from zero again. In CTC mode, the Timer/Counter will be cleared to zero when the Timer/Counter matches the given value.

### 8.2 Channels

Timer service has up to 3 channels output compare and 1 channel for input capture or overflow channel mode for 16bit timers. For 8bit timers, the Timer Service has 1 channel for output capture or overflow channel mode.

### 8.3 Interrupts/Channel Modes

Timer service supports 3 interrupt sources, Overflow, Output Compare, and Input Capture. However, only 16bit timers support Input Capture.

## 8.4 HalTimerInit ()

### 8.4.1 Description

This timer initialization function is called once at the startup. It should be called before any other timer function can be called. It allows hardware timers to be initialized with both required and optional parameters.

### 8.4.2 Prototype

```
void HalTimerInit (void)
```

### 8.4.3 Parameter Details

None.

### 8.4.4 Return

None.

## 8.5 HalTimerConfig()

### 8.5.1 Description

This function allows the channels to be configured in different modes.

### 8.5.2 Prototype

```
halTimerStatus_t HalTimerConfig ( uint8 timerId,
                                   uint8 opMode,
                                   uint8 channel,
                                   uint8 channelMode,
                                   bool intEnable,
                                   halTimerCBack_t cback);
```

### 8.5.3 Parameter Details

**timerId** - HAL\_TIMER\_X, identification of the timer. Different processors support different number of timers. ([See TimerId Table](#))

**opMode** – Operation Mode, can be Normal or CTC ([See Operation Modes Table](#))

**channel** – Channel to be configured. ([See Channels Table](#))

**channelMode** – The channel mode, Input Capture Mode, Output Compare Mode or Overflow Mode. ([See Channel Modes Table](#))

**intEnable** – TRUE or FALSE – Enable / Disable the interrupt

**cback** – pointer to the callback function. Callback function is used to inform the application when every there is an interrupt occurs.

```
typedef void (halTimerCback_t) (uint8 timerId,
                                uint8 channel,
                                uint8 channelMode);
```

**timerId** – HAL\_TIMER\_X, identification of the timer. Different processors support different number of timers. ([See TimerId Table](#))

**channel** – the channel where the interrupt occurs. (See Channels Table)

**channelMode** – Interrupt source of the event .

### 8.5.4 Return

Status of the configuration. ([See Status Table](#))

## 8.6 HalTimerStart ()

### 8.6.1 Description

This function starts the timer/counter with the operation mode, channel, channel mode, prescale that are provided by HalTimerConfig(). In other words, HalTimerConfig() has to be called before HalTimerStart() is called. In case of polling, timer ticks are updated by the Hal driver task calling HalTimerTick(). In case timer interrupt is used, timer ticks are updated by the interrupt every time an interrupt occurs.

### 8.6.2 Prototype

```
uint8 HalTimerStart ( uint8 timerId, uint32 timePerTick );
```

### 8.6.3 Parameter Details

**timerId** – HAL\_TIMER\_X, identification of the timer. Different processors support different number of timers. ([See TimerId Table](#))

**timerPerTick** – number of micro sec per tick

### 8.6.4 Return

If the Timer Service is not configured, HAL\_TIMER\_NOT\_CONFIGURED will be returned. Otherwise, HAL\_TIMER\_OK will be returned. ([See Status Table](#))

## 8.7 HalTimerStop()

### 8.7.1 Description

This function is called to stop a timer/counter.

### 8.7.2 Prototype

```
uint8 HalTimerStop (uint8 timerId);
```

### 8.7.3 Parameter Details

**timerId** – HAL\_TIMER\_X, identification of the timer that will be stopped. ([See Timer ID Table](#))

### 8.7.4 Return

If the timerId is not valid, HAL\_TIMER\_INVALID\_ID will be returned. Otherwise, HAL\_TIMER\_OK will be returned. ([See Status Table](#))

## 8.8 HalTimerTick()

### 8.8.1 Description

This function is called by the Hal Driver task when the interrupt is disabled in order to create a tick for the application. To use HalTimerTick(), Timer Service has to be configured using HalTimerConfig() with intEnable set to FALSE before calling HalTimerTick(). HalTimerTick() will send back to the application using the provided callback function at every tick. The duration of the tick is setup using the information provided by HalTimerConfig().

### 8.8.2 Prototype

```
void HalTimerTick (void);
```

### 8.8.3 Parameter Details

None.

### 8.8.4 Return

None.

## 8.9 HalTimerInterruptEnable()

### 8.9.1 Description

This function will enable, disable timer interrupt of the timerId and channelMode.

### 8.9.2 Prototype

```
uint8 HalTimerInterruptEnable ( uint8 timerId,
```

```
uint8 channelMode,
bool enable );
```

### 8.9.3 Parameter Details

**timerId** – HAL\_TIMER\_X, identification of the timer. Different processors support different number of timers. ([See TimerId Table](#))

**channelMode** – The channel mode, Input Capture Mode, Output Compare Mode or Overflow Mode. ([See Channel Modes Table](#))

**enable** – TRUE or FALSE

### 8.9.4 Return

If the timerId is not valid, HAL\_TIMER\_INVALID\_ID will be returned. Otherwise, HAL\_TIMER\_OK will be returned. ([See Status Table](#))

## 8.10 Constants

### 8.10.1 Timer ID

ID	Description
HAL_TIMER_0	8-bit timer ID
HAL_TIMER_1	16-bit timer ID - this is MAC timer and not supported by HAL
HAL_TIMER_2	8-bit timer ID
HAL_TIMER_3	16-bit timer ID

### 8.10.2 Channels

Channel	Description
HAL_TIMER_CHANNEL_SINGLE	Single Channel Timer
HAL_TIMER_CHANNEL_A	Timer Channel A
HAL_TIMER_CHANNEL_B	Timer Channel B
HAL_TIMER_CHANNEL_C	Timer Channel C

### 8.10.3 Channel Modes

Channel Mode	Description
HAL_TIMER_CH_MODE_INPUT_CAPTURE	Input Capture Mode
HAL_TIMER_CH_MODE_OUTPUT_COMPARE	Output Compare Mode
HAL_TIMER_CH_MODE_OVERFLOW	Overflow Mode



### 8.10.4 Operation Modes

Operation Mode	Description
HAL_TIMER_MODE_NORMAL	In Normal Mode, there is no counter clear performed. If the counter passes the max, it will restart again at zero
HAL_TIMER_MODE_CTC	In CTC Mode, the counter is cleared to zero when the counter matches with the specified value.

### 8.10.5 Prescale

Prescale – 8bit	Description
HAL_TIMER_8_TC_STOP	No clock, timer stopped
HAL_TIMER_8_TC_DIV1	No clock pre-scaling
HAL_TIMER_8_TC_DIV8	Clock pre-scaled by 8
HAL_TIMER_8_TC_DIV32	Clock pre-scaled by 32
HAL_TIMER_8_TC_DIV64	Clock pre-scaled by 64
HAL_TIMER_8_TC_DIV128	Clock pre-scaled by 128
HAL_TIMER_8_TC_DIV256	Clock pre-scaled by 256
HAL_TIMER_8_TC_DIV1024	Clock pre-scaled by 1024

Prescale – 16bit	Description
HAL_TIMER_16_TC_STOP	No clock, timer stopped
HAL_TIMER_16_TC_DIV1	No clock pre-scaling
HAL_TIMER_16_TC_DIV8	Clock pre-scaled by 8
HAL_TIMER_16_TC_DIV64	Clock pre-scaled by 64
HAL_TIMER_16_TC_DIV256	Clock pre-scaled by 256
HAL_TIMER_16_TC_DIV1024	Clock pre-scaled by 1024
HAL_TIMER_16_TC_EXTFE	External clock (T2), falling edge
HAL_TIMER_16_TC_EXTRE	External clock (T2), rising edge

### 8.10.6 Status

Status	Description
HAL_TIMER_OK	OK status

HAL_TIMER_NOT_OK	NOT OK status
HAL_TIMER_PARAMS_ERROR	Parameters are mismatched or no correct
HAL_TIMER_NOT_CONFIGURED	Timer is not configured
HAL_TIMER_INVALID_ID	Invalid Timer ID
HAL_TIMER_INVALID_CH_MODE	Invalid channel mode
HAL_TIMER_INVALID_OP_MODE	Invalid operation mode

## 9. UART Service

This service configures several things in the UART such as Baud rate, flow control, CTS, RTS, DSR, DTR, CD, RI...etc. It also reports framing and overrun errors.

### 9.1 HalUARTInit ()

#### 9.1.1 Description

This UART initialization function is called once at the startup. This function has to be called before any other UART function can be called. It enables UART to be initialized with both required and optional parameters.

#### 9.1.2 Prototype

```
void HalUARTInit (void)
```

#### 9.1.3 Parameter Details

None

#### 9.1.4 Return

None

### 9.2 HalUARTOpen ()

#### 9.2.1 Description

This function opens a port based on the configuration that is provided. A callback function is also registered so events can be handled correctly.

#### 9.2.2 Prototype

```
uint8 HalUARTOpen (uint8 port,
                   halUARTCfg_t *config);
```

#### 9.2.3 Parameter Details

**port** – specified serial port to be opened. ([Read UART Ports Table](#))

**config** – Structure that contains the information that is used to configure the port

```
typedef struct
{
    bool    configured;
```

```

uint16 baudRate;
bool    flowControl;
uint16 flowControlThreshold;
uint8   idleTimeout;
uint16 rx;
uint16 tx;
bool    intEnable;
uint32 rxChRvdTime;
halUARTCBack_t callbackFunc;
}halUARTCfg_t;

```

**config.configured** – Set by the function when the port is setup correctly and read to be used.

**config.baudRate** – The baud rate of the port to be opened. ([Check UART Ports Table](#))

**config.flowControl** – UART flow control can be set as TRUE or FALSE. TRUE value will enable flow control and FALSE value will disable flow control.

**config.flowControlThreshold** – Number of bytes left before Rx buffer reaches maxRxBufSize. When Rx buffer reaches this number (maxRxBufSize – flowControlThreshold) and flowControl is TRUE, a callback will be sent back to the application with HAL\_UART\_RX\_ABOUT\_FULL event.

**config.idleTimeout** – Rx timeout period in milliseconds. If Rx buffer haven't got new data for idleTimeout amount of time, a callback will be issued to the application with HAL\_UART\_RX\_TIMEOUT event. The application can choose to read everything from the Rx buffer or just partial of it.

**config.rx** – Contains halUARTBufControl\_t structure that used to manipulate Rx buffer

**config.tx** – Contains halUARTBufControl\_t structure that used to manipulate Tx buffer

```

typedef struct
{
    uint16 bufferHead;
    uint16 bufferTail;
    uint16 maxBufSize;
    uint8 *pBuffer;
}halUARTBufControl_t;

```

**bufferHead** – contain the index of the starting position of the Rx/Tx buffer

**bufferTail** – contains the index of the ending position of the Rx/Tx buffer

**maxBufSize** – holds maximum size of the Rx/Tx buffer that the UART can hold at a time. When this number is reached, HAL\_UART\_RX\_FULL or HAL\_UART\_TX\_FULL will be sent back to the application as an event through the callback system.

**\*pBuffer** – pointer to the buffer that contains the Rx data

**config.intEnable** – enable/disable interrupt. It can be set as TRUE or FALSE. TRUE value will enable the interrupt and FALSE value will disable the interrupt.

**callbackFunc** – This callback is called when there is an event such as Tx done, Rx ready...

```
void HalUARTCback (uint8 port, uint8 event);
```

**port** - specified serial port that has the event. ([Check UART Ports Table](#)).

**event** – event that causes the callback ([Check Events table](#)).

## 9.2.4 Return

Status of the function call. ([Check Status Table](#)).

## 9.3 HalUARTClose ()

### 9.3.1 Description

This function closes a given port. This allows settings to be reconfigured and opened again or power to be conserved by simply turn off the UART.

### 9.3.2 Prototype

```
void HalUARTClose (uint8 port);
```

### 9.3.3 Parameter Details

**port** - specified serial port to be closed. ([Check UART Ports Table](#))

### 9.3.4 Return

None.

## 9.4 HalUARTRead ()

### 9.4.1 Description

This function reads a buffer from the UART. The buffer must be allocated by the application using OSAL `osal_mem_alloc()`. The buffer then will be de-allocated by the application using `osal_mem_free()`. The number of bytes to be read is determined by the application. If the requested length is larger than the Rx Buffer, then the requested length will be adjusted to the Rx buffer length and the whole Rx buffer is return together with the adjusted length. If the requested length is smaller than the Rx buffer length, then only requested length buffer is sent back. The Rx buffer will be updated after the requested buffer is sent back. The function returns the length of the data if it is successful and 0 otherwise.

### 9.4.2 Prototype

```
uint16 HalUARTRead (uint8 port,  
                    uint8 *buf,  
                    uint16 length);
```

### 9.4.3 Parameter Details

**port** – specified serial port that data will be read. ([Check UART Ports Table](#))

**buf** – pointer to the buffer of the data. This buffer must be allocated using `osal_mem_alloc()` in the application and must be de-allocated after being used by the application using `osal_mem_free()`.

**length** – requested length.

#### 9.4.4 Return

Returns the length of the read data or 0 otherwise.

### 9.5 HalUARTWrite ()

#### 9.5.1 Description

This function writes a buffer of specific length into the serial port. The function will check if the Tx buffer is full or not. If the Tx Buffer is not full, the data will be loaded into the buffer and then will be sent to the Tx data register. If the Tx buffer is full, the function will send a callback HAL\_UART\_TX\_FULL and return 0. Otherwise, the length of the data that was sent will be returned.

#### 9.5.2 Prototype

```
uint16 HalUARTWrite (uint8 port,
                    uint8 *buf,
                    uint16 length);
```

#### 9.5.3 Parameter Details

**port** – specified serial port that data will be read. ([Check UART Ports Table](#))

**buf** – buffer of the data.

**length**– the length of the data

#### 9.5.4 Return

Returns the length of the data that is successfully written or 0 otherwise.

### 9.6 HalUARTIoctl ()

#### 9.6.1 Description

This function is used to perform operations on the port such as get/set, flush...etc

#### 9.6.2 Prototype

```
uint8 HalUARTIoctl (uint8 port,
                   uint8 cmd,
                   halUARTIoctl *ioctl);
```

#### 9.6.3 Parameter Details

**port** – specified serial port that data will be read. ([Check UART Ports Table](#))

**cmd** – operational commands

Command	Description
HAL_UART_IOCTL_SET	SET operation – use this to set the new setting for the UART
HAL_UART_IOCTL_GET	GET operation – use this to get the current setting of the UART
HAL_UART_IOCTL_FLUSH	FLUSH operation – use this to reset the current setting of the UART

**ioctl** – union of parameters associated with the commands

```
typedef union
{
    bool paramCTS;
    bool paramRTS;
    bool paramDSR;
    bool paramDTR;
    bool paramCD;
    bool paramRI;
    uint16 baudRate;
    bool flowControl;
    bool flushControl;
} halUARTIoctl;
```

**signals** – new setting for CTS, RTS, DSR, DTR, CD and RI.

**baudRate** – the new value of the baud rate for the given port. ([Check Baud Rate Table](#))

**flowControl** – UART flow control, can be set as TRUE or FALSE. TRUE value will enable flow control and FALSE value will disable flow control.

**flushControl** – UART flush control, can be set as TRUE or FALSE. TRUE value will stop all the operation and flush all the data.

#### 9.6.4 Return

Status of the function call. ([Check Status Table](#)).

### 9.7 HalUARTPoll()

#### 9.7.1 Description

This routine simulates the polling mechanism for UART.

#### 9.7.2 Prototype

```
void HalUARTPoll (void);
```

#### 9.7.3 Parameter Details

None.

#### 9.7.4 Return

None.

### 9.8 Hal\_UART\_RxBufLen()

#### 9.8.1 Description

This function returns the number of bytes currently in the Rx buffer.

#### 9.8.2 Prototype

```
uint16 Hal_UART_RxBufLen (uint8 port);
```

### 9.8.3 Parameter Details

**port** – specified serial port that length will be read. ([Check UART Ports Table](#))

### 9.8.4 Return

16 bit size of the Rx buffer.

## 9.9 Hal\_UART\_TxBufLen()

This function returns the number of bytes currently in the Tx buffer.

### 9.9.1 Description

This function returns the number of bytes currently in the Rx buffer.

### 9.9.2 Prototype

```
uint16 Hal_UART_TxBufLen (uint8 port);
```

### 9.9.3 Parameter Details

**port** – specified serial port that length will be read. ([Check UART Ports Table](#))

### 9.9.4 Return

16 bit size of the Tx buffer.

## 9.10 Hal\_UART\_FlowControlSet()

### 9.10.1 Description

This function enables or disables the flow control for the UART.

### 9.10.2 Prototype

```
void Hal_UART_SetFlowControl (uint8 port,  
                              uint8 status);
```

### 9.10.3 Parameter Details

**port** – specified serial port that flow control will be set. ([Check UART Ports Table](#))

**status** – TRUE or FALSE, enable or disable the flow control

### 9.10.4 Return

None.

## 9.11 Constants

### 9.11.1 UART Ports

Port	Description
HAL_UART_PORT_1	UART port 1
HAL_UART_PORT_2	UART port 2

### 9.11.2 Baud Rates

Parameter	Description
HAL_UART_BR_1200	Baud rate 1200 bps
HAL_UART_BR_2400	Baud rate 2400 bps
HAL_UART_BR_4800	Baud rate 4800 bps
HAL_UART_BR_9600	Baud rate 9600 bps
HAL_UART_BR_19200	Baud rate 19200 bps
HAL_UART_BR_31250	Baud rate 31250 bps
HAL_UART_BR_38400	Baud rate 38400 bps
HAL_UART_BR_57600	Baud rate 57600 bps
HAL_UART_BR_115200	Baud rate 115200 bps

### 9.11.3 Parity

Parameter	Description
HAL_UART_NO_PARITY	No parity
HAL_UART_ODD_PARITY	Odd parity
HAL_UART_EVEN_PARITY	Even parity

### 9.11.4 Stop Bits

Parameter	Description
HAL_UART_ONE_STOP_BIT	Stop Bits 1
HAL_UART_TWO_STOP_BITS	Stop Bits 2

### 9.11.5 Status

Parameter	Description
HAL_UART_SUCCESS	Success
HAL_UART_MEM_FAIL	Fail to allocate memory
HAL_UART_BAUDRATE_	Baud rate is bad

### 9.11.6 Callback Events



Event	Description
HAL_UART_RX_FULL	Rx Buffer is full
HAL_UART_RX_ABOUT_FULL	Rx Buffer is at <b>maxRxBufSize - flowControlThreshold</b>
HAL_UART_RX_TIMEOUT	Rx is idle for <b>idleTimeout</b> time
HAL_UART_TX_FULL	Tx Buffer is full