This document describes the API for the FT9XX Peripheral Driver Library.

Use of Bridgetek devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold Bridgetek harmless from any and all damages, claims, suits or expense resulting from such use.
# Table of Contents

1 Introduction ........................................................................................................... 7  
  1.1 Overview ........................................................................................................... 7  

2 Precompiled Libraries ........................................................................................... 8  
  2.1 Chip Management ............................................................................................. 8  
     2.1.1 API Cross Reference ............................................................................... 8  
     2.1.2 Enumeration Type .................................................................................... 9  
     2.1.3 Function Documentation ......................................................................... 11  
  2.2 Delay Functions ................................................................................................. 12  
     2.2.1 API Cross Reference ............................................................................... 12  
     2.2.2 Macro Definition Documentation ............................................................ 13  
  2.3 Interrupt Management ....................................................................................... 14  
     2.3.1 API Cross Reference ............................................................................... 14  
     2.3.2 Macro Definition Documentation ............................................................ 14  
     2.3.3 Typedef Documentation .......................................................................... 14  
     2.3.4 Enumeration Type Documentation .......................................................... 14  
     2.3.5 Function Documentation ......................................................................... 16  
  2.4 General Purpose I/O and Pad Control .............................................................. 18  
     2.4.1 API Cross Reference ............................................................................... 18  
     2.4.2 Function to Pad Mappings ........................................................................ 18  
     2.4.3 Enumeration Type Documentation .......................................................... 22  
     2.4.4 Function Documentation ......................................................................... 33  
  2.5 Assembler Definitions ....................................................................................... 38  
     2.5.1 API Cross Reference ............................................................................... 38  
     2.5.2 Macro Documentation .............................................................................. 38  
     2.5.3 Function Documentation ......................................................................... 42  
  2.6 Watchdog Timer ................................................................................................. 43  
     2.6.1 API Cross Reference ............................................................................... 43  
     2.6.2 Enumeration Type Documentation .......................................................... 43  
     2.6.3 Function Documentation ......................................................................... 44  
  2.7 Timers ............................................................................................................... 45
2.7.1 API Cross Reference ................................................................. 45
2.7.2 Enumeration Type Documentation ........................................... 45
2.7.3 Function Documentation .......................................................... 46

2.8 Analogue to Digital Converter ...................................................... 49
  2.8.1 API Cross Reference ............................................................ 49
  2.8.2 Enumeration Type Documentation ......................................... 49
  2.8.3 Function Documentation ....................................................... 50

2.9 Digital to Analogue Converter ..................................................... 53
  2.9.1 API Cross Reference ............................................................ 53
  2.9.2 Enumeration Type Documentation ......................................... 53
  2.9.3 Function Documentation ....................................................... 53

2.10 Ethernet driver ........................................................................ 57
  2.10.1 API Cross Reference ............................................................ 57
  2.10.2 Enumeration Type Documentation ......................................... 57
  2.10.3 Function Documentation ....................................................... 57

2.11 UART ......................................................................................... 61
  2.11.1 API Cross Reference ............................................................ 61
  2.11.2 Macro Definition Documentation ......................................... 61
  2.11.3 Enumeration Type Documentation ......................................... 62
  2.11.4 Function Documentation ....................................................... 64

2.12 I²C Master .................................................................................. 73
  2.12.1 API Cross Reference ............................................................ 73
  2.12.2 Enumeration Type Documentation ......................................... 73
  2.12.3 Function Documentation ....................................................... 74

2.13 I²C Slave .................................................................................... 77
  2.13.1 API Cross Reference ............................................................ 77
  2.13.2 Function Documentation ....................................................... 77

2.14 I²S Audio .................................................................................... 80
  2.14.1 API Cross Reference ............................................................ 80
  2.14.2 Enumeration Type Documentation ......................................... 80
  2.14.3 Function Documentation ....................................................... 83

2.15 SPI Bus ....................................................................................... 87
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.15.1</td>
<td>API Cross Reference</td>
<td>87</td>
</tr>
<tr>
<td>2.15.2</td>
<td>Enumeration Type Documentation</td>
<td>87</td>
</tr>
<tr>
<td>2.15.3</td>
<td>Function Documentation</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td><strong>CANBus</strong></td>
<td><strong>95</strong></td>
</tr>
<tr>
<td>2.16.1</td>
<td>API Cross Reference</td>
<td>95</td>
</tr>
<tr>
<td>2.16.2</td>
<td>Enumeration Type Documentation</td>
<td>95</td>
</tr>
<tr>
<td>2.16.3</td>
<td>Function Documentation</td>
<td>98</td>
</tr>
<tr>
<td>2.16.4</td>
<td>Variable Documentation</td>
<td>104</td>
</tr>
<tr>
<td>2.17</td>
<td><strong>Camera interface</strong></td>
<td><strong>105</strong></td>
</tr>
<tr>
<td>2.17.1</td>
<td>API Cross Reference</td>
<td>105</td>
</tr>
<tr>
<td>2.17.2</td>
<td>Enumeration Type Documentation</td>
<td>105</td>
</tr>
<tr>
<td>2.17.3</td>
<td>Function Documentation</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td><strong>Pulse Width Modulation</strong></td>
<td><strong>108</strong></td>
</tr>
<tr>
<td>2.18.1</td>
<td>API Cross Reference</td>
<td>108</td>
</tr>
<tr>
<td>2.18.2</td>
<td>Enumeration Type Documentation</td>
<td>108</td>
</tr>
<tr>
<td>2.18.3</td>
<td>Function Documentation</td>
<td>108</td>
</tr>
<tr>
<td>2.19</td>
<td><strong>PWM Audio</strong></td>
<td><strong>112</strong></td>
</tr>
<tr>
<td>2.19.1</td>
<td>API Cross Reference</td>
<td>112</td>
</tr>
<tr>
<td>2.19.2</td>
<td>Enumeration Type Documentation</td>
<td>112</td>
</tr>
<tr>
<td>2.19.3</td>
<td>Function Documentation</td>
<td>114</td>
</tr>
<tr>
<td>2.20</td>
<td><strong>Real Time Clock</strong></td>
<td><strong>117</strong></td>
</tr>
<tr>
<td>2.20.1</td>
<td>FT90X and FT93X register definitions</td>
<td>117</td>
</tr>
<tr>
<td>2.20.2</td>
<td>Enumeration Type Documentation</td>
<td>117</td>
</tr>
<tr>
<td>2.20.3</td>
<td>Function Documentation</td>
<td>118</td>
</tr>
<tr>
<td>2.21</td>
<td><strong>USB Device Stack API</strong></td>
<td><strong>122</strong></td>
</tr>
<tr>
<td>2.21.1</td>
<td>API Cross Reference</td>
<td>122</td>
</tr>
<tr>
<td>2.21.2</td>
<td>Macro Definition Documentation</td>
<td>123</td>
</tr>
<tr>
<td>2.21.3</td>
<td>TypeDefinition Documentation</td>
<td>123</td>
</tr>
<tr>
<td>2.21.4</td>
<td>Enumeration Type Documentation</td>
<td>124</td>
</tr>
<tr>
<td>2.21.5</td>
<td>Structure Documentation</td>
<td>127</td>
</tr>
<tr>
<td>2.21.6</td>
<td>Function Documentation</td>
<td>129</td>
</tr>
<tr>
<td>2.22</td>
<td><strong>USB Device Stack Extensions API</strong></td>
<td><strong>137</strong></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>2.22.1</td>
<td>API Cross Reference</td>
<td>137</td>
</tr>
<tr>
<td>2.22.2</td>
<td>Structure Documentation</td>
<td>137</td>
</tr>
<tr>
<td>2.22.3</td>
<td>Function Documentation</td>
<td>138</td>
</tr>
<tr>
<td>2.23</td>
<td>DFU Device for USB Device Stack API</td>
<td>141</td>
</tr>
<tr>
<td>2.23.1</td>
<td>API Cross Reference</td>
<td>141</td>
</tr>
<tr>
<td>2.23.2</td>
<td>Macro Definition Documentation</td>
<td>141</td>
</tr>
<tr>
<td>2.23.3</td>
<td>Function Documentation</td>
<td>142</td>
</tr>
<tr>
<td>2.24</td>
<td>High Bandwidth Isochronous IN support in USB Device Stack API</td>
<td>145</td>
</tr>
<tr>
<td>2.24.1</td>
<td>API Cross Reference</td>
<td>145</td>
</tr>
<tr>
<td>2.24.2</td>
<td>Macro Definition Documentation</td>
<td>145</td>
</tr>
<tr>
<td>2.24.3</td>
<td>Enumeration Type Documentation</td>
<td>145</td>
</tr>
<tr>
<td>2.24.4</td>
<td>Function Documentation</td>
<td>146</td>
</tr>
<tr>
<td>2.25</td>
<td>USB Host Stack API</td>
<td>147</td>
</tr>
<tr>
<td>2.25.1</td>
<td>API Cross Reference</td>
<td>147</td>
</tr>
<tr>
<td>2.25.2</td>
<td>Macro Definition Documentation</td>
<td>147</td>
</tr>
<tr>
<td>2.25.3</td>
<td>Typedef Documentation</td>
<td>148</td>
</tr>
<tr>
<td>2.25.4</td>
<td>Structure Documentation</td>
<td>149</td>
</tr>
<tr>
<td>2.25.5</td>
<td>Enumeration Type Documentation</td>
<td>151</td>
</tr>
<tr>
<td>2.25.6</td>
<td>Function Documentation</td>
<td>153</td>
</tr>
<tr>
<td>2.26</td>
<td>USB Host Stack Extensions API</td>
<td>168</td>
</tr>
<tr>
<td>2.26.1</td>
<td>API Cross Reference</td>
<td>168</td>
</tr>
<tr>
<td>2.26.2</td>
<td>Function Documentation</td>
<td>168</td>
</tr>
<tr>
<td>2.27</td>
<td>HID Devices on USB Host Stack API</td>
<td>170</td>
</tr>
<tr>
<td>2.27.1</td>
<td>API Cross Reference</td>
<td>170</td>
</tr>
<tr>
<td>2.27.2</td>
<td>Structure Documentation</td>
<td>171</td>
</tr>
<tr>
<td>2.27.3</td>
<td>Function Documentation</td>
<td>171</td>
</tr>
<tr>
<td>2.28</td>
<td>BOMS Devices on USB Host Stack API</td>
<td>174</td>
</tr>
<tr>
<td>2.28.1</td>
<td>API Cross Reference</td>
<td>174</td>
</tr>
<tr>
<td>2.28.2</td>
<td>Macro Definition Documentation</td>
<td>174</td>
</tr>
<tr>
<td>2.28.3</td>
<td>Structure Documentation</td>
<td>175</td>
</tr>
<tr>
<td>2.28.4</td>
<td>Function Documentation</td>
<td>176</td>
</tr>
<tr>
<td>2.29</td>
<td>CDC ACM Devices on USB Host Stack API</td>
<td>180</td>
</tr>
</tbody>
</table>
2.29.1 Macro Definition Documentation ........................................... 181
2.29.2 Structure Documentation......................................................... 182
2.29.3 Function Documentation.......................................................... 184

2.30 Android Open Accessory (AOA) Devices on USB Host Stack API 191
2.30.1 API Cross Reference.................................................................. 191
2.30.2 Macro Definition Documentation .............................................. 191
2.30.3 Structure Documentation........................................................... 192
2.30.4 Function Documentation........................................................... 193

2.31 FT devices on USB host stack API (ft900_usbh_ft.h) ....197
2.31.1 API Cross Reference.................................................................. 197
2.31.2 Structure Documentation........................................................... 197
2.31.3 Functions.................................................................................. 198

2.32 Startup DFU Feature .................................................................202
2.32.1 API Cross Reference.................................................................. 202
2.32.2 Macro Definition Documentation .............................................. 203
2.32.3 Function Documentation........................................................... 203

2.33 SD Host.......................................................................................203
2.33.1 Enumeration Type Documentation............................................. 203
2.33.2 Function Documentation........................................................... 205

2.34 Datalogger Feature.......................................................................206
2.34.1 Datalogger Partition ................................................................. 206
2.34.2 API Cross Reference.................................................................. 207
2.34.3 Variable Documentation........................................................... 207
2.34.4 Function Documentation........................................................... 207

2.35 D2XX Feature..............................................................................208
2.35.1 API Cross Reference.................................................................. 209
2.35.2 Variable Documentation........................................................... 209
2.35.3 Macro Definition Documentation .............................................. 209
2.35.4 Structure Documentation........................................................... 210
2.35.5 Enumeration Type Documentation............................................. 211
2.35.6 Typedef Documentation............................................................ 212
2.35.7 Function Documentation........................................................... 212
3 Header Files................................................................. 216
  3.1 Hardware Register Definition Files............................ 216
    3.1.1 Using Register Header Files.................................. 216
  3.2 API Header Files.................................................. 218
  3.3 Additional Header Files........................................ 219
4 Contact Information ............................................. 220
Appendix A – References ........................................... 221
  Document References ............................................... 221
  Acronyms and Abbreviations ..................................... 221
Appendix B – List of Tables & Figures ............................ 223
  List of Tables............................................................ 223
  List of Figures.......................................................... 223
Appendix C – Revision History ...................................... 224
1 Introduction

The FT9XX Peripheral Driver Library is a collection of ‘C’ language based functions that are intended to ease the development of applications running on the FT90X or FT93X Microcontroller.

Figure 1-1 and Figure 1-2 shows the overall FT9XX peripherals driver support. This document focuses on the Hardware Interface Drivers layer. All drivers will be provided as source code for easy adaptation and modification.

Figure 1-1 FT90X Peripherals Driver Support

Figure 1-2 FT93X Interface Driver Support

This document will describe the APIs for the FT9XX Peripheral Driver Library.
2 Precompiled Libraries

The precompiled libraries provided with the FT9XX Toolchain are shown in Error! Reference source not found.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libft900.a</td>
<td>Peripheral driver library for FT90X Series of MCUs</td>
</tr>
<tr>
<td>libft900_d2xx_dev.a</td>
<td>D2XX library for FT90X Series of MCUs</td>
</tr>
<tr>
<td>libft900_d2xx_dev_rtos.a</td>
<td>D2XX library for FT90X Series of MCUs with support for FreeRTOS</td>
</tr>
<tr>
<td>libft930.a</td>
<td>Peripheral driver library for FT93X Series of MCUs</td>
</tr>
<tr>
<td>libft930_d2xx_dev.a</td>
<td>D2XX library for FT93X Series of MCUs [Note that FT93X does not require a special library for FreeRTOS. This library can be used for both RTOS and non-RTOS use cases]</td>
</tr>
<tr>
<td>libftd2xx_host.a</td>
<td>D2XX Host library for FT90X Series of MCUs</td>
</tr>
</tbody>
</table>

All libraries are built in two modes – Debug and Release. Debug uses –Og optimization while Release uses –O0. The libraries are located in the toolchain installation folder at the relative path Toolchain\hardware\lib\Debug and Toolchain\hardware\lib\Release.

The precompiled driver libraries can be used as is. The source code to the library is provided and it may be modified. To change the source code, make a local copy of the source code pertaining to the module into the Eclipse project. The linker command line ensures that the local copy of the API object is used during linking.

For example, if you want to force the Ethernet to use 10 Mbit/sec mode only, then copy the source code file, ethernet.c to the project and make the required changes to ETHERNET_AUTO_NEG_ALLOW and ETHERNET_MODE macro definitions in ethernet.c, in the project’s workspace.

Compiling against the library will take the local version in preference to the library’s version. Source code can be found here (once IDE has been installed) at the relative path: Toolchain\hardware\src

The sources for the D2xx libraries are not released with the toolchain. Please contact support@brtchip.com if access to the source code is required.

2.1 Chip Management

The file ft900_sys.h contains the definitions for the chip management functions in the libft900.a library and libft930.a

2.1.1 API Cross Reference

It utilises the following library APIs:

- **ft900_delay.h** – Delay

Additional definitions are taken from:

- **ft900_registers.h** – FT90x and FT93x register definitions
2.1.2 Enumeration Type

2.1.2.1 sys_device_t

enum sys_device_t
FT90x Devices.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_device_usb_host</td>
<td>USB Host</td>
</tr>
<tr>
<td>sys_device_usb_device</td>
<td>USBDevice</td>
</tr>
<tr>
<td>sys_device_ethernet</td>
<td>Ethernet</td>
</tr>
<tr>
<td>sys_device_sd_card</td>
<td>SD Card</td>
</tr>
<tr>
<td>sys_device_can0</td>
<td>CAN0</td>
</tr>
<tr>
<td>sys_device_can1</td>
<td>CAN1</td>
</tr>
<tr>
<td>sys_device_i2c_master</td>
<td>I2C Master</td>
</tr>
<tr>
<td>sys_device_i2c_slave</td>
<td>I2C Slave</td>
</tr>
<tr>
<td>sys_device_spi_master</td>
<td>SPI Master</td>
</tr>
<tr>
<td>sys_device_spi_slave0</td>
<td>SPI Slave 0</td>
</tr>
<tr>
<td>sys_device_spi_slave1</td>
<td>SPI Slave 1</td>
</tr>
<tr>
<td>sys_device_uart0</td>
<td>UART0</td>
</tr>
<tr>
<td>sys_device_uart1</td>
<td>UART1</td>
</tr>
<tr>
<td>sys_device_pwm</td>
<td>PWM</td>
</tr>
<tr>
<td>sys_device_i2s</td>
<td>I2S</td>
</tr>
<tr>
<td>sys_device_camera</td>
<td>Camera</td>
</tr>
<tr>
<td>sys_device_timer_wdt</td>
<td>Timer and Watchdog Timer</td>
</tr>
<tr>
<td>sys_device_adc</td>
<td>Analogue to Digital Converter</td>
</tr>
<tr>
<td>sys_device_dac0</td>
<td>Digital to Analogue Converter 0</td>
</tr>
<tr>
<td>sys_device_dac1</td>
<td>Digital to Analogue Converter 1</td>
</tr>
</tbody>
</table>

FT93x Devices.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_device_uart2</td>
<td>UART2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>sys_device_uart3</td>
<td>UART3</td>
</tr>
<tr>
<td>sys_device_pwm</td>
<td>PWM</td>
</tr>
<tr>
<td>sys_device_uart1</td>
<td>UART1</td>
</tr>
<tr>
<td>sys_device_uart0</td>
<td>UART0</td>
</tr>
<tr>
<td>sys_device_spi_slave0</td>
<td>SPI Slave</td>
</tr>
<tr>
<td>sys_device_spi_master</td>
<td>SPI Master</td>
</tr>
<tr>
<td>sys_device_i2c_slave</td>
<td>I2C Slave</td>
</tr>
<tr>
<td>sys_device_spi_master</td>
<td>SPI Master</td>
</tr>
<tr>
<td>sys_device_i2c_master</td>
<td>I2C Master</td>
</tr>
<tr>
<td>sys_device_spi_master</td>
<td>SPI Master</td>
</tr>
<tr>
<td>sys_device_i2c_master</td>
<td>I2C Master</td>
</tr>
<tr>
<td>sys_device_usb_device</td>
<td>USB Device</td>
</tr>
<tr>
<td>sys_device_timer_wdt</td>
<td>Timer and Watchdog Timer</td>
</tr>
<tr>
<td>sys_device_adc</td>
<td>ADC</td>
</tr>
<tr>
<td>sys_device_dac0</td>
<td>DAC0</td>
</tr>
<tr>
<td>sys_device_dac1</td>
<td>DAC1</td>
</tr>
</tbody>
</table>

**2.1.2.2 sys_cpu_divider_t**

`enum sys_cpu_divider_t`

CPU Clock divider.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_cpu_divider_1</td>
<td>No clock divider (Default)</td>
</tr>
<tr>
<td>sys_cpu_divider_2</td>
<td>Divide Input Clock by 2</td>
</tr>
<tr>
<td>sys_cpu_divider_4</td>
<td>Divide Input Clock by 4</td>
</tr>
<tr>
<td>sys_cpu_divider_8</td>
<td>Divide Input Clock by 8</td>
</tr>
<tr>
<td>sys_cpu_divider_64</td>
<td>Divide Input Clock by 64</td>
</tr>
<tr>
<td>sys_cpu_divider_128</td>
<td>Divide Input Clock by 128</td>
</tr>
<tr>
<td>sys_cpu_divider_512</td>
<td>Divide Input Clock by 512</td>
</tr>
</tbody>
</table>

**2.1.2.3 sys_pwm_trigger_t**

`enum sys_pwm_trigger_t`

PWM External Trigger pin (only for FT90x).
## Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_pwm_trigger_none</td>
<td>None</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio18</td>
<td>GPIO18</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio26</td>
<td>GPIO26</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio35</td>
<td>GPIO35</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio40</td>
<td>GPIO40</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio46</td>
<td>GPIO46</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio52</td>
<td>GPIO52</td>
</tr>
<tr>
<td>sys_pwm_trigger_gpio58</td>
<td>GPIO58</td>
</tr>
</tbody>
</table>

### 2.1.3 Function Documentation

#### 2.1.3.1 sys_enable

```c
int sys_enable ( sys_device_t dev )
```

Enable a device on the FT9xx.

**Parameters**

- `dev` The device to enable

**Returns**

On success a 0, otherwise -1

#### 2.1.3.2 sys_disable

```c
int sys_disable ( sys_device_t dev )
```

Disable a device on the FT9xx.

**Parameters**

- `dev` The device to Disable

**Returns**

On success a 0, otherwise -1

#### 2.1.3.3 sys_reset_all

```c
void sys_reset_all ( void )
```

Reset all peripherals. `sys_cpu_clock_div`

**Parameters**

- `div` The divider to use
Returns
On success a 0, otherwise -1

2.1.3.4 sys_get_cpu_clock

uint32_t sys_get_cpu_clock ( void )

Get the current clock of the CPU.

Returns
The clock rate of the CPU in Hertz

2.1.3.5 sys_i2c_swop

int sys_i2c_swop ( uint8_t swop )

Swap the I2C Master and slave pins. The user must first configure the default master pins and assign the swapped I2C master to those pins. [This function is only available for FT90X] For example:

    gpio_function(44, pad_i2c1_scl);
    gpio_pull(44, pad_pull_none);
    gpio_function(45, pad_i2c1_sda);
    gpio_pull(45, pad_pull_none);

Parameters

    swop Enable or disable the swop feature

Returns
On success a 0, otherwise -1

2.1.3.6 sys_pwm_ext_trigger

int sys_pwm_ext_trigger ( sys_pwm_trigger_t extrigger )

Configure the External PWM trigger. [This function is only available for FT90X]

Parameters

    extrigger The selection of external trigger

Returns
On success a 0, otherwise -1

2.1.3.7 sys_check_ft900_revB

Function macro that checks whether the revision of FT90X series is Revision B.

Returns
True if device is Revision B and False is returned if FT900 Revision C or FT93x.

2.2 Delay Functions

The file ft900_delay.h contains the definitions for the delay functions in the libft900.a and libft930.a libraries.

2.2.1 API Cross Reference

Additional definitions are taken from:
2.2.2 Macro Definition Documentation

2.2.2.1 sleep

#define sleep (x) delayms(x*1000)

POSIX standard second sleep call.

**Note:** This function consists of a tight loop counting CPU cycles to perform the delay. It is not recommended to use this function call at interrupt level or in FreeRTOS applications.

**Parameters**

- `x` The number of milliseconds to sleep

2.2.2.2 usleep

#define usleep (x) delayus(x)

POSIX standard microsecond sleep call.

**Note:** This function consists of a tight loop counting CPU cycles to perform the delay. It is not recommended to use this function call at interrupt level or in FreeRTOS applications.

**Parameters**

- `x` The number of microseconds to sleep
2.3 Interrupt Management

The file ft900_interrupt.h contains the definitions for the interrupt management functions in the libft900.a and libft930.a libraries.

2.3.1 API Cross Reference

Additional definitions are taken from:

*ft900_registers.h* – FT90X and FT93X register definitions

2.3.2 Macro Definition Documentation

2.3.2.1 N_INTERRUPTS

```c
#define N_INTERRUPTS (34)
```

The number of interrupts supported by the CPU, includes watch dog interrupt vector which is not under the purview of interrupt controller.

2.3.3 Typedef Documentation

2.3.3.1 isrptr_t

```c
typedef void(* isrptr_t) (void)
```

Interrupt handler function prototype.

2.3.4 Enumeration Type Documentation

2.3.4.1 interrupt_t

```c
eenum interrupt_t
```

FT90X Interrupt vectors:

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_0</td>
<td>Reserved</td>
</tr>
<tr>
<td>interrupt_usb_host</td>
<td>USB Host Interrupt</td>
</tr>
<tr>
<td>interrupt_usb_device</td>
<td>USB Device Interrupt</td>
</tr>
<tr>
<td>interrupt_ethernet</td>
<td>Ethernet Interrupt</td>
</tr>
<tr>
<td>interrupt_sd_card</td>
<td>SD Card Interrupt</td>
</tr>
<tr>
<td>interrupt_can0</td>
<td>CAN0 Interrupt</td>
</tr>
<tr>
<td>interrupt_can1</td>
<td>CAN1 Interrupt</td>
</tr>
<tr>
<td>interrupt_camera</td>
<td>Camera Interrupt</td>
</tr>
<tr>
<td>interrupt_spim</td>
<td>SPI Master Interrupt</td>
</tr>
<tr>
<td>interrupt_spis0</td>
<td>SPI Slave 0 Interrupt</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>interrupt_spis1</td>
<td>SPI Slave 1 Interrupt</td>
</tr>
<tr>
<td>interrupt_i2cm</td>
<td>I2C Master Interrupt</td>
</tr>
<tr>
<td>interrupt_i2cs</td>
<td>I2C Slave Interrupt</td>
</tr>
<tr>
<td>interrupt_uart0</td>
<td>UART0 Interrupt</td>
</tr>
<tr>
<td>interrupt_uart1</td>
<td>UART1 Interrupt</td>
</tr>
<tr>
<td>interrupt_i2s</td>
<td>I2S Interrupt</td>
</tr>
<tr>
<td>interrupt_pwm</td>
<td>PWM Interrupt</td>
</tr>
<tr>
<td>interrupt_timers</td>
<td>Timers Interrupt</td>
</tr>
<tr>
<td>interrupt_gpio</td>
<td>GPIO Interrupt</td>
</tr>
<tr>
<td>interrupt_RTC</td>
<td>RTC Interrupt</td>
</tr>
<tr>
<td>interrupt_adc</td>
<td>ADC Interrupt</td>
</tr>
<tr>
<td>interrupt_dac</td>
<td>DAC Interrupt</td>
</tr>
<tr>
<td>interrupt_slowclock</td>
<td>Slow clock timer interrupt</td>
</tr>
<tr>
<td>interrupt_wdg</td>
<td>First level watchdog timeout interrupt</td>
</tr>
</tbody>
</table>

FT93x Interrupt vectors:

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_0</td>
<td>Reserved</td>
</tr>
<tr>
<td>interrupt_1</td>
<td>Reserved</td>
</tr>
<tr>
<td>interrupt_usb_device</td>
<td>USB Device Interrupt</td>
</tr>
<tr>
<td>interrupt_3</td>
<td>Reserved</td>
</tr>
<tr>
<td>interrupt_sd_card</td>
<td>SD Card Interrupt</td>
</tr>
<tr>
<td>Interrupt_mailbox_source</td>
<td>Mailbox Source Interrupt</td>
</tr>
<tr>
<td>interrupt_mailbox_dest</td>
<td>Mailbox Destination Interrupt</td>
</tr>
<tr>
<td>interrupt_uart3</td>
<td>UART3 Interrupt</td>
</tr>
<tr>
<td>interrupt_spim</td>
<td>SPI Master Interrupt</td>
</tr>
<tr>
<td>interrupt_spis0</td>
<td>SPI Slave 0 Interrupt</td>
</tr>
<tr>
<td>interrupt_10</td>
<td>Reserved</td>
</tr>
<tr>
<td>interrupt_i2cm</td>
<td>I2C Master Interrupt</td>
</tr>
<tr>
<td>interrupt_i2cs</td>
<td>I2C Slave Interrupt</td>
</tr>
<tr>
<td>interrupt</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>interrupt_uart0</td>
<td>UART0 Interrupt</td>
</tr>
<tr>
<td>interrupt_uart1</td>
<td>UART1 Interrupt</td>
</tr>
<tr>
<td>interrupt_uart2</td>
<td>UART2 Interrupt</td>
</tr>
<tr>
<td>interrupt_pwm</td>
<td>PWM Interrupt</td>
</tr>
<tr>
<td>interrupt_timers</td>
<td>Timers Interrupt</td>
</tr>
<tr>
<td>interrupt_gpio</td>
<td>GPIO Interrupt</td>
</tr>
<tr>
<td>interrupt RTC</td>
<td>RTC Interrupt</td>
</tr>
<tr>
<td>interrupt_adc</td>
<td>ADC Interrupt</td>
</tr>
<tr>
<td>interrupt_dac</td>
<td>DAC Interrupt</td>
</tr>
<tr>
<td>interrupt_slowclock</td>
<td>Slow Clock Timer</td>
</tr>
<tr>
<td>interrupt_7channel_fifo</td>
<td>7 Channel FIFO interrupt</td>
</tr>
<tr>
<td>interrupt_wdg</td>
<td>First level watchdog timeout interrupt</td>
</tr>
</tbody>
</table>

### 2.3.5 Function Documentation

#### 2.3.5.1 interrupt_attach

```c
int8_t interrupt_attach ( interrupt_t interrupt,
                          uint8_t priority,
                          isrptr_t func
)                                      
```

Attach an interrupt.

**Parameters**

- `interrupt` The interrupt vector to attach to
- `priority` The priority to give the interrupt.
- `func` The function to call when interrupted

**Returns**

0 on a success or -1 for a failure

**Note:** `interrupt_attach` for a peripheral interrupt should be called prior to enabling that peripheral's interrupt. Doing otherwise could lead to a system hang

#### 2.3.5.2 interrupt_detach

```c
int8_t interrupt_detach ( interrupt_t interrupt )
```

Detach an interrupt.

**Parameters**
### interrupt
The interrupt vector to detach

**Returns**
0 on a success or -1 for a failure

#### 2.3.5.3 interrupt_disable_globally

```c
int8_t interrupt_disable_globally ( void )
```
Disable all interrupts.

**Returns**
0 on a success or -1 for a failure

#### 2.3.5.4 interrupt_disable_nesting

```c
int8_t interrupt_disable_nesting ( void )
```
Disable nesting interrupts.

**Returns**
0 on a success or -1 for a failure

#### 2.3.5.5 interrupt_enable_globally

```c
int8_t interrupt_enable_globally ( void )
```
Enable interrupts to fire.

**Returns**
0 on a success or -1 for a failure

#### 2.3.5.6 interrupt_enable_nesting

```c
int8_t interrupt_enable_nesting ( uint8_t max )
```
Enable nesting interrupts.

**Parameters**
- `max` The maximum number of levels to nest (max 16)

**Returns**
0 on a success or -1 for a failure
2.4 General Purpose I/O and Pad Control

The file `ft900_gpio.h` contains the definitions for the GPIO and Pad Control functions in the `libft900.a` and `libft930.a` libraries.

2.4.1 API Cross Reference

Additional definitions are taken from:

`ft900_registers.h` – FT90X and FT93X register definitions

2.4.2 Function to Pad Mappings

Pins on FT90X and FT93X have multiple functions mapped onto them. The required function is selected by configuring the pin to its corresponding pad function. `pad_func_x (X=0 to 3)` select the mapping. The available functions on a pin are shown in the following table.

<table>
<thead>
<tr>
<th>Pin</th>
<th>pad_func_0</th>
<th>pad_func_1</th>
<th>pad_func_2</th>
<th>pad_func_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBUS_DISCH/GPIO0</td>
<td>GPIO0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC_N/GPIO1</td>
<td>GPIO1</td>
<td>OC_N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSW_N/GPIO2</td>
<td>GPIO2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBUS_DTC/GPIO3</td>
<td>GPIO3</td>
<td>VBUS_DTC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENET_LED0/GPIO4</td>
<td>GPIO4</td>
<td>ENET_LED0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENET_LED1/GPIO5</td>
<td>GPIO5</td>
<td>ENET_LED1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC1/CAM_XCLK/GPIO6</td>
<td>GPIO6</td>
<td>CAM_XCLK</td>
<td>ADC1</td>
<td></td>
</tr>
<tr>
<td>ADC2/CAM_PCLK/GPIO7</td>
<td>GPIO7</td>
<td>CAM_PCLK</td>
<td>ADC2</td>
<td></td>
</tr>
<tr>
<td>ADC3/CAM_VD/GPIO8</td>
<td>GPIO8</td>
<td>CAM_VD</td>
<td>ADC3</td>
<td></td>
</tr>
<tr>
<td>ADC4/CAM_HD/GPIO9</td>
<td>GPIO9</td>
<td>CAM_HD</td>
<td>ADC4</td>
<td></td>
</tr>
<tr>
<td>ADC5/CAM_D7/GPIO10</td>
<td>GPIO10</td>
<td>CAM_D7</td>
<td>ADC5</td>
<td></td>
</tr>
<tr>
<td>ADC6/CAM_D6/GPIO11</td>
<td>GPIO11</td>
<td>CAM_D6</td>
<td>ADC6</td>
<td></td>
</tr>
<tr>
<td>ADC7/CAM_D5/GPIO12</td>
<td>GPIO12</td>
<td>CAM_D5</td>
<td>ADC7</td>
<td></td>
</tr>
<tr>
<td>DAC1/CAM_D4/GPIO13</td>
<td>GPIO13</td>
<td>CAM_D4</td>
<td>DAC1</td>
<td></td>
</tr>
<tr>
<td>DAC0/CAM_D3/GPIO14</td>
<td>GPIO14</td>
<td>CAM_D3</td>
<td>DAC0</td>
<td></td>
</tr>
<tr>
<td>CAN0_TXD/CAM_D2/GPIO15</td>
<td>GPIO15</td>
<td>CAM_D2</td>
<td>CAN0_TXD</td>
<td></td>
</tr>
<tr>
<td>CAN0_RXD/CAM_D1/GPIO16</td>
<td>GPIO16</td>
<td>CAM_D1</td>
<td>CAN0_RXD</td>
<td></td>
</tr>
<tr>
<td>CAN1_TXD/CAM_D0/GPIO17</td>
<td>GPIO17</td>
<td>CAM_D0</td>
<td>CAN1_TXD</td>
<td></td>
</tr>
<tr>
<td>CAN1_RXD/GPIO18</td>
<td>GPIO18</td>
<td></td>
<td>CAN1_RXD</td>
<td></td>
</tr>
<tr>
<td>SD_CLK/GPIO19</td>
<td>GPIO19</td>
<td>SD_CLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_CMD/GPIO20</td>
<td>GPIO20</td>
<td>SD_CMD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_DATA3/GPIO21</td>
<td>GPIO21</td>
<td>SD_DATA3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin</td>
<td>pad_func_0</td>
<td>pad_func_1</td>
<td>pad_func_2</td>
<td>pad_func_3</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>SD_DATA2/GPIO22</td>
<td>GPIO22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_DATA1/GPIO23</td>
<td>GPIO23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_DATA0/GPIO24</td>
<td>GPIO24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_CD/GPIO25</td>
<td>GPIO25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_WP/GPIO26</td>
<td>GPIO26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_CLK/GPIO27</td>
<td>GPIO27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SSO/GPIO28</td>
<td>GPIO28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_MOSI/GPIO29</td>
<td>GPIO29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_MISO/GPIO30</td>
<td>GPIO30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_IO2/GPIO31</td>
<td>GPIO31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_IO3/GPIO32</td>
<td>GPIO32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SS1/GPIO33</td>
<td>GPIO33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SS2/GPIO34</td>
<td>GPIO34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SS3/GPIO35</td>
<td>GPIO35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIS0_CLK/GPIO36</td>
<td>GPIO36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPISO_SS/GPIO37</td>
<td>GPIO37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPISO_MOSI/GPIO38</td>
<td>GPIO38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPISO_MISO/GPIO39</td>
<td>GPIO39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIS1_CLK/GPIO40</td>
<td>GPIO40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIS1_SS/GPIO41</td>
<td>GPIO41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIS1_MOSI/GPIO42</td>
<td>GPIO42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIS1_MISO/GPIO43</td>
<td>GPIO43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2C0_SCL/GPIO44</td>
<td>GPIO44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2C0_SDA/GPIO45</td>
<td>GPIO45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2C1_SCL/GPIO46</td>
<td>GPIO46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2C1_SDA/GPIO47</td>
<td>GPIO47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_TXD/GPIO48</td>
<td>GPIO48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_RXD/GPIO49</td>
<td>GPIO49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_RTS/GPIO50</td>
<td>GPIO50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_CTS/GPIO51</td>
<td>GPIO51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_DTR/UART1_TXD/PWM4/GPIO52</td>
<td>GPIO52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_DSR/UART1_RXD/PWM5/GPIO53</td>
<td>GPIO53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART0_DCD/UART1_RTS/PWM6/GPIO54</td>
<td>GPIO54</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3 - FT93X Pin Mapping

<table>
<thead>
<tr>
<th>Pin</th>
<th>pad_func_0</th>
<th>pad_func_1</th>
<th>pad_func_2</th>
<th>pad_func_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART0_RI/UART1_CTS/PWM7/GPIO55</td>
<td>GPIO55</td>
<td>PWM7</td>
<td>UART1_CTS</td>
<td>UART0_RI</td>
</tr>
<tr>
<td>PWM0/GPIO56</td>
<td>GPIO56</td>
<td>PWM0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM1/GPIO57</td>
<td>GPIO57</td>
<td>PWM1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM2/GPIO58</td>
<td>GPIO58</td>
<td>PWM2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM3/GPIO59</td>
<td>GPIO59</td>
<td>PWM3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2S_SDAO/GPIO60</td>
<td>GPIO60</td>
<td>I2S_SDAO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2S_SDAI/GPIO61</td>
<td>GPIO61</td>
<td>I2S_SDAI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2S_BCLK/GPIO62</td>
<td>GPIO62</td>
<td>I2S_BCLK</td>
<td>I2S_BCLK</td>
<td></td>
</tr>
<tr>
<td>I2S_LRCLK/GPIO63</td>
<td>GPIO63</td>
<td>I2S_LRCLK</td>
<td>I2S_LRCLK</td>
<td></td>
</tr>
<tr>
<td>I2S_MCLK/GPIO64</td>
<td>GPIO64</td>
<td>I2S_MCLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2S_CLK22/GPIO65</td>
<td>GPIO65</td>
<td>I2S_CLK22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2S_CLK24/GPIO66</td>
<td>GPIO66</td>
<td>I2S_CLK24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD_CLK/SPIS_CLK/GPIO0</td>
<td>GPIO0</td>
<td>SPI_SCLK</td>
<td>SD_CLK</td>
<td></td>
</tr>
<tr>
<td>SD_CMD/SPIS_MISO/GPIO1</td>
<td>GPIO1</td>
<td>SPI_SMISO</td>
<td>SD_CMD</td>
<td></td>
</tr>
<tr>
<td>SD_CD/SPIS_MOSI/GPIO2</td>
<td>GPIO2</td>
<td>SPI_SMOI</td>
<td>SD_CD</td>
<td></td>
</tr>
<tr>
<td>SD_DATA0/SPIS_SS/GPIO3</td>
<td>GPIO3</td>
<td>SPI_SS</td>
<td>SD_DATA0</td>
<td></td>
</tr>
<tr>
<td>PWM0/SD_DATA1/PWM7/GPIO4</td>
<td>GPIO4</td>
<td>PWM7</td>
<td>SD_DATA1</td>
<td>PWM0</td>
</tr>
<tr>
<td>PWM6/SD_DATA2/PWM1/GPIO5</td>
<td>GPIO5</td>
<td>PWM1</td>
<td>SD_DATA2</td>
<td>PWM6</td>
</tr>
<tr>
<td>SD_DATA3/PWM5/GPIO6</td>
<td>GPIO6</td>
<td>PWM5</td>
<td>SD_DATA3</td>
<td></td>
</tr>
<tr>
<td>SD_WP/PWM4/GPIO7</td>
<td>GPIO7</td>
<td>PWM4</td>
<td>SD_WP</td>
<td></td>
</tr>
<tr>
<td>PWM3/GPIO8</td>
<td>GPIO8</td>
<td>PWM3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM2/GPIO9</td>
<td>GPIO9</td>
<td>PWM2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM1/GPIO10</td>
<td>GPIO10</td>
<td>PWM1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM0/GPIO11</td>
<td>GPIO11</td>
<td>PWM0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2CS_SCL/I2CM_SCL/GPIO12</td>
<td>GPIO12</td>
<td>I2CS_SCL</td>
<td>I2CS_SCL</td>
<td></td>
</tr>
<tr>
<td>I2CS_SDA/I2CM_SDA/GPIO13</td>
<td>GPIO13</td>
<td>I2CS_SDA</td>
<td>I2CS_SDA</td>
<td></td>
</tr>
<tr>
<td>UART2_RXD/GPIO14</td>
<td>GPIO14</td>
<td>UART2_RXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART2_TXD/GPIO15</td>
<td>GPIO15</td>
<td>UART2_TXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART2_RTS/GPIO16</td>
<td>GPIO16</td>
<td>UART2_RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin</td>
<td>pad_func_0</td>
<td>pad_func_1</td>
<td>pad_func_2</td>
<td>pad_func_3</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>UART2_CTS/GPIO17</td>
<td>GPIO17</td>
<td>UART2_CTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART3_RXD_UART2_DTR/GPIO18</td>
<td>GPIO18</td>
<td>UART2_DTR</td>
<td>UART3_RXD</td>
<td></td>
</tr>
<tr>
<td>UART3_TXD_UART2_DSR/GPIO19</td>
<td>GPIO19</td>
<td>UART2_DSR</td>
<td>UART3_TXD</td>
<td></td>
</tr>
<tr>
<td>UART3_RTS_UART2_DCD/GPIO20</td>
<td>GPIO20</td>
<td>UART2_DCD</td>
<td>UART3_RTS</td>
<td></td>
</tr>
<tr>
<td>UART3_CTS_UART2_RI/GPIO21</td>
<td>GPIO21</td>
<td>UART2_RI</td>
<td>UART3_CTS</td>
<td></td>
</tr>
<tr>
<td>PWM3_UART0_RXD/GPIO22</td>
<td>GPIO22</td>
<td>UART0_RXD</td>
<td>PWM3</td>
<td></td>
</tr>
<tr>
<td>PWM2_UART0_TXD/GPIO23</td>
<td>GPIO23</td>
<td>UART0_TXD</td>
<td>PWM2</td>
<td></td>
</tr>
<tr>
<td>PWM1_UART0_RTS/GPIO24</td>
<td>GPIO24</td>
<td>UART0_RTS</td>
<td>PWM1</td>
<td></td>
</tr>
<tr>
<td>PWM0_UART0_CTS/GPIO25</td>
<td>GPIO25</td>
<td>UART0_CTS</td>
<td>PWM0</td>
<td></td>
</tr>
<tr>
<td>UART1_RXD_UART0_DTR/GPIO26</td>
<td>GPIO26</td>
<td>UART0_DTR</td>
<td>UART1_RXD</td>
<td></td>
</tr>
<tr>
<td>UART1_TXD_UART0_DSR/GPIO27</td>
<td>GPIO27</td>
<td>UART0_DSR</td>
<td>UART1_TXD</td>
<td></td>
</tr>
<tr>
<td>UART1_RTS_UART0_DCD/GPIO28</td>
<td>GPIO28</td>
<td>UART0_DCD</td>
<td>UART1_RTS</td>
<td></td>
</tr>
<tr>
<td>SPIM_SS0_UART1_CTS_UART0_RI/GPIO29</td>
<td>GPIO29</td>
<td>UART0_RI</td>
<td>UART1_CTS</td>
<td>SPIM_SS0</td>
</tr>
<tr>
<td>SPIM_SS0/GPIO30</td>
<td>GPIO30</td>
<td>SPIM_SS0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SS1/GPIO31</td>
<td>GPIO31</td>
<td>SPIM_SS1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SS2/GPIO32</td>
<td>GPIO32</td>
<td>SPIM_SS2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIM_SS3/GPIO33</td>
<td>GPIO33</td>
<td>SPIM_SS3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPI5_CLK_SPI5_CLK/GPIO34</td>
<td>GPIO34</td>
<td>SPI5_CLK</td>
<td>SPI5_CLK</td>
<td></td>
</tr>
<tr>
<td>SPI5_MISO_SPI5_MISO/GPIO35</td>
<td>GPIO35</td>
<td>SPI5_MISO</td>
<td>SPI5_MISO</td>
<td></td>
</tr>
<tr>
<td>SPI5_MOSI_SPI5_MOSI/GPIO36</td>
<td>GPIO36</td>
<td>SPI5_MOSI</td>
<td>SPI5_MOSI</td>
<td></td>
</tr>
<tr>
<td>SPI5_SS_SPI5_IO2/GPIO37</td>
<td>GPIO37</td>
<td>SPI5_IO2</td>
<td>SPI5_SS</td>
<td></td>
</tr>
<tr>
<td>RTC_REF_SPI5_IO3/GPIO38</td>
<td>GPIO38</td>
<td>SPI5_IO3</td>
<td>RTC_REF</td>
<td></td>
</tr>
<tr>
<td>SPI50_MISO/GPIO39</td>
<td>GPIO39</td>
<td>SPI50_MISO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBUS_DTC/GPIO39</td>
<td>GPIO40</td>
<td>VBUS_DTC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.4.3 Enumeration Type Documentation

2.4.3.1 gpio_int_edge_t

```c
enum gpio_int_edge_t
  GPIO Interrupt control.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio_int_edge_falling</td>
<td>Interrupt on a falling edge</td>
</tr>
<tr>
<td>gpio_int_edge_raising</td>
<td>Interrupt on a raising edge</td>
</tr>
</tbody>
</table>

2.4.3.2 pad_dir_t

```c
def enum pad_dir_t
  Pad direction control.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_dir_input</td>
<td>Input</td>
</tr>
<tr>
<td>pad_dir_output</td>
<td>Output</td>
</tr>
<tr>
<td>pad_dir_open_drain</td>
<td>Open Drain</td>
</tr>
</tbody>
</table>

2.4.3.3 pad_drive_t

```c
def enum pad_drive_t
  Pad current drive control.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_drive_4mA</td>
<td>4mA maximum current</td>
</tr>
<tr>
<td>pad_drive_8mA</td>
<td>8mA maximum current</td>
</tr>
<tr>
<td>pad_drive_12mA</td>
<td>12mA maximum current</td>
</tr>
<tr>
<td>pad_drive_16mA</td>
<td>16mA maximum current</td>
</tr>
</tbody>
</table>

2.4.3.4 pad_func_t

```c
def enum pad_func_t
  Pad function control for FT90X
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_func_0</td>
<td>Pad function 0</td>
</tr>
<tr>
<td>pad_func_1</td>
<td>Pad function 1</td>
</tr>
<tr>
<td>pad_func_2</td>
<td>Pad function 2</td>
</tr>
<tr>
<td>pad_func_3</td>
<td>Pad function 3</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>pad_gpio0</td>
<td>VBUS_DISCH/GPIO0 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio1</td>
<td>OC_N/GPIO1 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio2</td>
<td>PSW_N/GPIO2 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio3</td>
<td>VBUS_DTC/GPIO3 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio4</td>
<td>ENET_LED0/GPIO4 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio5</td>
<td>ENET_LED1/GPIO5 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio6</td>
<td>ADC1/CAM_XCLK/GPIO6 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio7</td>
<td>ADC2/CAM_PCLK/GPIO7 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio8</td>
<td>ADC3/CAM_VD/GPIO8 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio9</td>
<td>ADC4/CAM_HD/GPIO9 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio10</td>
<td>ADC5/CAM_D7/GPIO10 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio11</td>
<td>ADC6/CAM_D6/GPIO11 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio12</td>
<td>ADC7/CAM_D5/GPIO12 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio13</td>
<td>DAC1/CAM_D4/GPIO13 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio14</td>
<td>DAC0/CAM_D3/GPIO14 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio15</td>
<td>CAN0_TXD/CAM_D2/GPIO15 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio16</td>
<td>CAN0_RXD/CAM_D1/GPIO16 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio17</td>
<td>CAN1_TXD/CAM_D0/GPIO17 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio18</td>
<td>CAN1_RXD/GPIO18 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio19</td>
<td>SD_CLK/GPIO19 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio20</td>
<td>SD_CMD/GPIO20 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio21</td>
<td>SD_DATA3/GPIO21 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio22</td>
<td>SD_DATA2/GPIO22 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio23</td>
<td>SD_DATA1/GPIO23 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio24</td>
<td>SD_DATA0/GPIO24 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio25</td>
<td>SD_CD/GPIO25 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio26</td>
<td>SD_WP/GPIO26 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio27</td>
<td>SPI_CLK/GPIO27 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio28</td>
<td>SPIM_SS0/GPIO28 Pad function 0</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>pad_gpio29</td>
<td>SPIM_MOSI/GPIO29 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio30</td>
<td>SPIM_MISO/GPIO30 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio31</td>
<td>SPIM_IO2/GPIO31 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio32</td>
<td>SPIM_IO3/GPIO32 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio33</td>
<td>SPIM_SS1/GPIO33 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio34</td>
<td>SPIM_SS2/GPIO34 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio35</td>
<td>SPIM_SS3/GPIO35 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio36</td>
<td>SPIS0_CLK/GPIO36 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio37</td>
<td>SPIS0_SS/GPIO37 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio38</td>
<td>SPIS0_MOSI/GPIO38 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio39</td>
<td>SPIS0_MISO/GPIO39 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio40</td>
<td>SPIS1_CLK/GPIO40 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio41</td>
<td>SPIS1_SS/GPIO41 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio42</td>
<td>SPIS1_MOSI/GPIO42 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio43</td>
<td>SPIS1_MISO/GPIO43 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio44</td>
<td>I2C0_SCL/GPIO44 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio45</td>
<td>I2C0_SDA/GPIO45 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio46</td>
<td>I2C1_SCL/GPIO46 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio47</td>
<td>I2C1_SDA/GPIO47 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio48</td>
<td>UART0_TXD/GPIO48 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio49</td>
<td>UART0_RXD/GPIO49 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio50</td>
<td>UART0_RTS/GPIO50 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio51</td>
<td>UART0_CTS/GPIO51 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio52</td>
<td>UART0_DTR/UART1_TXD/PWM4/GPIO52 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio53</td>
<td>UART0_DSR/UART1_RXD/PWM5/GPIO53 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio54</td>
<td>UART0_DCD/UART1_RTS/PWM6/GPIO54 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio55</td>
<td>UART0_RI/UART1_CTS/PWM7/GPIO55 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio56</td>
<td>PWM0/GPIO56 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio57</td>
<td>PWM1/GPIO57 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio58</td>
<td>PWM2/GPIO58 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio59</td>
<td>PWM3/GPIO59 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio60</td>
<td>I2S_SDAO/GPIO60 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio61</td>
<td>I2S_SDAI/GPIO61 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio62</td>
<td>I2S_BCLK/GPIO62 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio63</td>
<td>I2S_LRCLK/GPIO63 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio64</td>
<td>I2S_MCLK/GPIO64 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio65</td>
<td>I2S_CLK22/GPIO65 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio66</td>
<td>I2S_CLK24/GPIO66 Pad function 0</td>
</tr>
<tr>
<td>pad_oc_n</td>
<td>OC_N/GPIO1 Pad function 1</td>
</tr>
<tr>
<td>pad_vbus_dtc</td>
<td>VBUS_DTC/GPIO3 Pad function 1</td>
</tr>
<tr>
<td>pad_enet_led0</td>
<td>ENET_LED0/GPIO4 Pad function 1</td>
</tr>
<tr>
<td>pad_enet_led1</td>
<td>ENET_LED1/GPIO5 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_xclk</td>
<td>ADC1/CAM_XCLK/GPIO6 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_pclk</td>
<td>ADC2/CAM_PCLK/GPIO7 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_vd</td>
<td>ADC3/CAM_VD/GPIO8 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_hd</td>
<td>ADC4/CAM HD/GPIO9 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d7</td>
<td>ADC5/CAM_D7/GPIO10 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d6</td>
<td>ADC6/CAM_D6/GPIO11 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d5</td>
<td>ADC7/CAM_D5/GPIO12 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d4</td>
<td>DAC1/CAM_D4/GPIO13 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d3</td>
<td>DAC0/CAM_D3/GPIO14 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d2</td>
<td>CAN0_TXD/CAM_D2/GPIO15 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d1</td>
<td>CAN0_RXD/CAM_D1/GPIO16 Pad function 1</td>
</tr>
<tr>
<td>pad_cam_d0</td>
<td>CAN1_TXD/CAM_D0/GPIO17 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_clk</td>
<td>SD_CLK/GPIO19 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_cmd</td>
<td>SD_CMD/GPIO20 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_data3</td>
<td>SD_DATA3/GPIO21 Pad function 1</td>
</tr>
<tr>
<td>Pad Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>pad_sd_data2</td>
<td>SD_DATA2/GPIO22 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_data1</td>
<td>SD_DATA1/GPIO23 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_data0</td>
<td>SD_DATA0/GPIO24 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_cd</td>
<td>SD_CD/GPIO25 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_wp</td>
<td>SD_WP/GPIO26 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_sck</td>
<td>SPIM_CLK/GPIO27 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_ss0</td>
<td>SPIM_SS0/GPIO28 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_mosi</td>
<td>SPIM_MOSI/GPIO29 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_miso</td>
<td>SPIM_MISO/GPIO30 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_io2</td>
<td>SPIM_IO2/GPIO31 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_io3</td>
<td>SPIM_IO3/GPIO32 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_ss1</td>
<td>SPIM_SS1/GPIO33 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_ss2</td>
<td>SPIM_SS2/GPIO34 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_ss3</td>
<td>SPIM_SS3/GPIO35 Pad function 1</td>
</tr>
<tr>
<td>pad_spis0_clk</td>
<td>SPIS0_CLK/GPIO36 Pad function 1</td>
</tr>
<tr>
<td>pad_spis0_ss</td>
<td>SPIS0_SS/GPIO37 Pad function 1</td>
</tr>
<tr>
<td>pad_spis0_mosi</td>
<td>SPIS0_MOSI/GPIO38 Pad function 1</td>
</tr>
<tr>
<td>pad_spis0_miso</td>
<td>SPIS0_MISO/GPIO39 Pad function 1</td>
</tr>
<tr>
<td>pad_spis1_clk</td>
<td>SPIS1_CLK/GPIO40 Pad function 1</td>
</tr>
<tr>
<td>pad_spis1_ss</td>
<td>SPIS1_SS/GPIO41 Pad function 1</td>
</tr>
<tr>
<td>pad_spis1_mosi</td>
<td>SPIS1_MOSI/GPIO42 Pad function 1</td>
</tr>
<tr>
<td>pad_spis1_miso</td>
<td>SPIS1_MISO/GPIO43 Pad function 1</td>
</tr>
<tr>
<td>pad_i2c0_scl</td>
<td>I2C0_SCL/GPIO44 Pad function 1</td>
</tr>
<tr>
<td>pad_i2c0_sda</td>
<td>I2C0_SDA/GPIO45 Pad function 1</td>
</tr>
<tr>
<td>pad_i2c1_scl</td>
<td>I2C1_SCL/GPIO46 Pad function 1</td>
</tr>
<tr>
<td>pad_i2c1_sda</td>
<td>I2C1_SDA/GPIO47 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm4</td>
<td>UART0_DTR/UART1_TXD/PWM4/GPIO52 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm5</td>
<td>UART0_DSR/UART1_RXD/PWM5/GPIO53 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm6</td>
<td>UART0_DCD/UART1_RTS/PWM6/GPIO54 Pad function 1</td>
</tr>
<tr>
<td>Pad Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>pad_pwm7</td>
<td>UART0_RI/UART1_CTS/PWM7/GPIO55 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm0</td>
<td>PWM0/GPIO56 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm1</td>
<td>PWM1/GPIO57 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm2</td>
<td>PWM2/GPIO58 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm3</td>
<td>PWM3/GPIO59 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_sdao</td>
<td>I2S_SDAO/GPIO60 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_sdaip</td>
<td>I2S_SDAI/GPIO61 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_bclk</td>
<td>I2S_BCLK/GPIO62 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_lclk</td>
<td>I2S_LRCLK/GPIO63 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_mclk</td>
<td>I2S_MCLK/GPIO64 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_clk22</td>
<td>I2S_CLK22/GPIO65 Pad function 1</td>
</tr>
<tr>
<td>pad_i2s_clk24</td>
<td>I2S_CLK24/GPIO66 Pad function 1</td>
</tr>
<tr>
<td>pad_can0_txd</td>
<td>CAN0_TXD/CAN_D2/GPIO15 Pad function 2</td>
</tr>
<tr>
<td>pad_can0_rxd</td>
<td>CAN0_RXD/CAN_D1/GPIO16 Pad function 2</td>
</tr>
<tr>
<td>pad_can1_txd</td>
<td>CAN1_TXD/CAN_D0/GPIO17 Pad function 2</td>
</tr>
<tr>
<td>pad_can1_rxd</td>
<td>CAN1_RXD/GPIO18 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_txd</td>
<td>UART0_DTR/UART1_TXD/PWM4/GPIO52 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_rxd</td>
<td>UART0_DSR/UART1_RXD/PWM5/GPIO53 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_rts</td>
<td>UART0_DCD/UART1_RTS/PWM6/GPIO54 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_cts</td>
<td>UART0_RI/UART1_CTS/PWM7/GPIO55 Pad function 2</td>
</tr>
<tr>
<td>pad_i2ss_bclk</td>
<td>I2S_BCLK/GPIO62 Pad function 2</td>
</tr>
<tr>
<td>pad_i2ss_lclk</td>
<td>I2S_LRCLK/GPIO63 Pad function 2</td>
</tr>
<tr>
<td>pad_adc1</td>
<td>ADC1/CAM_XCLK/GPIO6 Pad function 3</td>
</tr>
<tr>
<td>pad_adc2</td>
<td>ADC2/CAM_PCLK/GPIO7 Pad function 3</td>
</tr>
<tr>
<td>pad_adc3</td>
<td>ADC3/CAM_VD/GPIO8 Pad function 3</td>
</tr>
<tr>
<td>pad_adc4</td>
<td>ADC4/CAM_HD/GPIO9 Pad function 3</td>
</tr>
<tr>
<td>pad_adc5</td>
<td>ADC5/CAM_D7/GPIO10 Pad function 3</td>
</tr>
<tr>
<td>pad_adc6</td>
<td>ADC6/CAM_D6/GPIO11 Pad function 3</td>
</tr>
<tr>
<td>pad_adc7</td>
<td>ADC7/CAM_D5/GPIO12 Pad function 3</td>
</tr>
</tbody>
</table>
### Pad function control for FT93x

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_func_0</td>
<td>Pad function 0</td>
</tr>
<tr>
<td>pad_func_1</td>
<td>Pad function 1</td>
</tr>
<tr>
<td>pad_func_2</td>
<td>Pad function 2</td>
</tr>
<tr>
<td>pad_func_3</td>
<td>Pad function 3</td>
</tr>
<tr>
<td>pad_gpio0</td>
<td>SD_CLK/SPIS_CLK/GPIO0 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio1</td>
<td>SD_CMD/SPIS_MISO/GPIO1 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio2</td>
<td>SD_CD/SPIS_MOSI/GPIO2 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio3</td>
<td>SD_DATA0/SPIS_SS/GPIO3 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio4</td>
<td>PWM0/SD_DATA1/PWM7/GPIO4 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio5</td>
<td>PWM6/SD_DATA2/PWM1/GPIO5 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio6</td>
<td>SD_DATA3/PWM5/GPIO6 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio7</td>
<td>SD_WP/PWM4/GPIO7 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio8</td>
<td>PWM3/GPIO8 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio9</td>
<td>PWM2/GPIO9 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio10</td>
<td>PWM1/GPIO10 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio11</td>
<td>PWM0/GPIO11 Pad function 0</td>
</tr>
<tr>
<td>Pad Name</td>
<td>Function Details</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>pad_gpio12</td>
<td>I2CS_SCL/I2CM_SCL/GPIO12 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio13</td>
<td>I2CS_SDA/I2CM_SDA/GPIO13 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio14</td>
<td>UART2_RXD/GPIO14 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio15</td>
<td>UART2_TXD/GPIO15 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio16</td>
<td>UART2_RTS/GPIO16 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio17</td>
<td>UART2_CTS/GPIO17 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio18</td>
<td>UART3_RXD/UART2_DTR/GPIO18 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio19</td>
<td>UART3_TXD/UART2_DSR/GPIO19 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio20</td>
<td>UART3_RTS/UART2_DCD/GPIO20 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio21</td>
<td>UART3_CTS/UART2_RI/GPIO21 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio22</td>
<td>PWM3/UART0_RXD/GPIO22 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio23</td>
<td>PWM2/UART0_TXD/GPIO23 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio24</td>
<td>PWM1/UART0_RTS/GPIO24 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio25</td>
<td>PWM0/UART0_CTS/GPIO25 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio26</td>
<td>UART1_RXD/UART0_DTR/GPIO26 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio27</td>
<td>UART1_TXD/UART0_DSR/GPIO27 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio28</td>
<td>UART1_RTS/UART0_DCD/GPIO28 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio29</td>
<td>SPIM_SS0/UART1_CTS/UART0_RI/GPIO29 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio30</td>
<td>SPIM_SS0/GPIO30 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio31</td>
<td>SPIM_SS1/GPIO31 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio32</td>
<td>SPIM_SS2/GPIO32 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio33</td>
<td>SPIM_SS3/GPIO33 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio34</td>
<td>SPIS_CLK/SPIM_CLK/GPIO34 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio35</td>
<td>SPIS_MISO/SPIM_MISO/GPIO35 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio36</td>
<td>SPIS_MOSI/SPIM_MOSI/GPIO36 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio37</td>
<td>SPIS_SS/SPIM_IO2/GPIO37 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio38</td>
<td>RTC_REF/SPIM_IO3/GPIO38 Pad function 0</td>
</tr>
<tr>
<td>pad_gpio39</td>
<td>VBUS_DTC/GPIO39 Pad function 0</td>
</tr>
<tr>
<td>pad0_spis0_clk</td>
<td>SD_CLK/SPIS_CLK/GPIO0 Pad function 1</td>
</tr>
</tbody>
</table>
### FT9XX API Programmers Manual

<table>
<thead>
<tr>
<th>Pad Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad1_spis0_miso</td>
<td>SD_CMD/SPIS_MISO/GPIO1 Pad function 1</td>
</tr>
<tr>
<td>pad2_spis0_mosi</td>
<td>SD_CD/SPIS_MOSI/GPIO2 Pad function 1</td>
</tr>
<tr>
<td>pad3_spis0_ss</td>
<td>SD_DATA0/SPIS_SS/GPIO3 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm7</td>
<td>PWM0/SD_DATA1/PWM7/GPIO4 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm6</td>
<td>PWM6/SD_DATA2/PWM1/GPIO5 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm5</td>
<td>SD_DATA3/PWM5/GPIO6 Pad function 1</td>
</tr>
<tr>
<td>pad_pwm4</td>
<td>SD_WP/PWM4/GPIO7 Pad function 1</td>
</tr>
<tr>
<td>pad8_pwm3</td>
<td>PWM3/GPIO8 Pad function 1</td>
</tr>
<tr>
<td>pad9_pwm2</td>
<td>PWM2/GPIO9 Pad function 1</td>
</tr>
<tr>
<td>pad11_pwm0</td>
<td>PWM0/GPIO11 Pad function 1</td>
</tr>
<tr>
<td>pad10_pwm1</td>
<td>PWM1/GPIO10 Pad function 1</td>
</tr>
<tr>
<td>pad_i2cm_scl</td>
<td>I2CS_SCL/I2CM_SCL/GPIO12 Pad function 1</td>
</tr>
<tr>
<td>pad_i2cm_sda</td>
<td>I2CS_SDA/I2CM_SDA/GPIO13 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_rxd</td>
<td>UART2_RXD/GPIO14 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_txd</td>
<td>UART2_TXD/GPIO15 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_rts</td>
<td>UART2_RTS/GPIO16 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_cts</td>
<td>UART2_CTS/GPIO17 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_dtr</td>
<td>UART3_RXD/UART2_DTR/GPIO18 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_dsr</td>
<td>UART3_TXD/UART2_DSR/GPIO19 Pad function 1</td>
</tr>
<tr>
<td>pad_uart2_dcd</td>
<td>UART3_RTS/UART2_DCD/GPIO20 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_rxd</td>
<td>UART3_CTS/UART2_RI/GPIO21 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_rts</td>
<td>PWM3/UART0_RXD/GPIO22 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_cts</td>
<td>PWM2/UART0_TXD/GPIO23 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_RDS</td>
<td>PWM1/UART0_RTS/GPIO24 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_dtr</td>
<td>PWM0/UART0_CTS/GPIO25 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_dsr</td>
<td>UART1_RXD/UART0_DTR/GPIO26 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0_dcd</td>
<td>UART1_TXD/UART0_DSR/GPIO27 Pad function 1</td>
</tr>
<tr>
<td>pad_uart0 Ri</td>
<td>UART1_RTS/UART0_DCD/GPIO28 Pad function 1</td>
</tr>
<tr>
<td>pad30_spism_ss0</td>
<td>SPIM_SS0/UART1_CTS/UART0_RI/GPIO29 Pad function 1</td>
</tr>
<tr>
<td>Pad Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>pad_spim_ss1</td>
<td>SPIM_SS1/GPIO31 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_ss2</td>
<td>SPIM_SS2/GPIO32 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_ss3</td>
<td>SPIM_SS3/GPIO33 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_sck</td>
<td>SPIM_CLK/SPIS_CLK/GPIO34 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_miso</td>
<td>SPIS_MISO/SPIM_MISO/GPIO35 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_mosi</td>
<td>SPIS_MOSI/SPIM_MOSI/GPIO36 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_io2</td>
<td>SPIS_SS/SPIM_IO2/GPIO37 Pad function 1</td>
</tr>
<tr>
<td>pad_spim_io3</td>
<td>RTC_REF/SPIM_IO3/GPIO38 Pad function 1</td>
</tr>
<tr>
<td>pad_vbus_dtc</td>
<td>VBUS_DTC/GPIO39 Pad function 1</td>
</tr>
<tr>
<td>pad_sd_clk</td>
<td>SD_CLK/SPIS_CLK/GPIO0 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_cmd</td>
<td>SD_CMD/SPIS_MISO/GPIO1 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_cd</td>
<td>SD_CD/SPIS_MOSI/GPIO2 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_data0</td>
<td>SD_DATA0/SPIS_SS/GPIO3 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_data1</td>
<td>PWM0/SD_DATA1/PWM7/GPIO4 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_data2</td>
<td>PWM6/SD_DATA2/PWM1/GPIO5 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_data3</td>
<td>SD_DATA3/PWM5/GPIO6 Pad function 2</td>
</tr>
<tr>
<td>pad_sd_wp</td>
<td>SD_WP/PWM4/GPIO7 Pad function 2</td>
</tr>
<tr>
<td>pad_i2cs_scl</td>
<td>I2CS_SCL/I2CM_SCL/GPIO12 Pad function 2</td>
</tr>
<tr>
<td>pad_i2cs_sda</td>
<td>I2CS_SDA/I2CM_SDA/GPIO13 Pad function 2</td>
</tr>
<tr>
<td>pad_uart3_rxd</td>
<td>UART3_RXD/UART2_DTR/GPIO18 Pad function 2</td>
</tr>
<tr>
<td>pad_uart3_txd</td>
<td>UART3_TXD/UART2_DSR/GPIO19 Pad function 2</td>
</tr>
<tr>
<td>pad_uart3_rts</td>
<td>UART3_RTS/UART2_DCD/GPIO20 Pad function 2</td>
</tr>
<tr>
<td>pad_uart3_cts</td>
<td>UART3_CTS/UART2_RI/GPIO21 Pad function 2</td>
</tr>
<tr>
<td>pad22_pwm3</td>
<td>PWM3/UART0_RXD/GPIO22 Pad function 1</td>
</tr>
<tr>
<td>pad23_pwm2</td>
<td>PWM2/UART0_TXD/GPIO23 Pad function 1</td>
</tr>
<tr>
<td>pad24_pwm1</td>
<td>PWM1/UART0_RTS/GPIO24 Pad function 2</td>
</tr>
<tr>
<td>pad25_pwm0</td>
<td>PWM0/UART0_CTS/GPIO25 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_txd</td>
<td>UART1_RXD/UART0_DTR/GPIO26 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_rxd</td>
<td>UART1_TXD/UART0_DSR/GPIO27 Pad function 2</td>
</tr>
<tr>
<td>pad_uart1_rts</td>
<td>UART1_RTS/UART0_DCD/GPIO28 Pad function 2</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>pad_uart1_cts</td>
<td>SPIM_SS0/UART1_CTS/UART0_RI/GPIO29 Pad function 2</td>
</tr>
<tr>
<td>pad_spis0_clk</td>
<td>SPIM_CLK/SPIS_CLK/GPIO34 Pad function 2</td>
</tr>
<tr>
<td>pad_spis0_miso</td>
<td>SPIS_MISO/SPIM_MISO/GPIO35 Pad function 2</td>
</tr>
<tr>
<td>pad_spis0_mosi</td>
<td>SPIS_MOSI/SPIM_MOSI/GPIO36 Pad function 2</td>
</tr>
<tr>
<td>pad_spis0_ss</td>
<td>SPIS_SS/SPIM_IO2/GPIO37 Pad function 2</td>
</tr>
<tr>
<td>pad_rtc_ref</td>
<td>RTC_REF/SPIM_IO3/GPIO38 Pad function 2</td>
</tr>
<tr>
<td>pad29_spim_ss0</td>
<td>SPIM_SS0/UART1_CTS/UART0_RI/GPIO29 Pad function 3</td>
</tr>
<tr>
<td>pad4_pwm0</td>
<td>PWM0/SD_DATA1/PWM7/GPIO4 Pad function 3</td>
</tr>
<tr>
<td>pad5_pwm1</td>
<td>PWM1/SD_DATA2/PWM6/GPIO5 Pad function 3</td>
</tr>
</tbody>
</table>
2.4.3.5  **pad_pull_t**

```c
enum pad_pull_t
```

Pad pull up and pull downs control.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_pull_none</td>
<td>No pull up or pull down</td>
</tr>
<tr>
<td>pad_pull_pullup</td>
<td>Weak pull up enabled</td>
</tr>
<tr>
<td>pad_pull_pulldown</td>
<td>Weak pull down enabled</td>
</tr>
<tr>
<td>pad_pull_keeper</td>
<td>Weak pull up/down reflects output</td>
</tr>
</tbody>
</table>

2.4.3.6  **pad_schmitt_t**

```c
enum pad_schmitt_t
```

Pad Schmitt trigger control.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_schmitt_off</td>
<td>Pad input is filtered through a schmitt trigger</td>
</tr>
<tr>
<td>pad_schmitt_on</td>
<td>Pad input is unfiltered</td>
</tr>
</tbody>
</table>

2.4.3.7  **pad_slew_t**

```c
enum pad_slew_t
```

Pad slew rate control.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_slew_fast</td>
<td>Fast Slew Rate</td>
</tr>
<tr>
<td>pad_slew_slow</td>
<td>Slow Slew Rate</td>
</tr>
</tbody>
</table>

2.4.4  **Function Documentation**

2.4.4.1  **gpio_dir**

```c
int8_t gpio_dir ( uint8_t num, pad_dir_t dir )
```

Configure the direction of a pin.

**Parameters**

- **num** The GPIO number
- **dir** The direction
Returns
On success a 0, otherwise -1

2.4.4.2 gpio_function

```c
int8_t gpio_function ( uint8_t num,
                        pad_func_t func
                   )
```
Configure the alternative function for a pin.

Parameters
- **num** The GPIO number
- **func** The function that the pin should use

Returns
On success a 0, otherwise -1

2.4.4.3 gpio_idrive

```c
int8_t gpio_idrive ( uint8_t num,
                        pad_drive_t drive
                   )
```
Configure the maximum current drive for a pin.

Parameters
- **num** The GPIO number
- **drive** The maximum current

Returns
On success a 0, otherwise -1

2.4.4.4 gpio_pull

```c
int8_t gpio_pull ( uint8_t num,
                        pad_pull_t pull
                   )
```
Configure the pull up/down for a pin.

Parameters
- **num** The GPIO number
- **pull** The pullup/down configuration

Returns
On success a 0, otherwise -1
2.4.4.5 gpio_schmitt

```c
int8_t gpio_schmitt ( uint8_t num, 
    pad_schmitt_t schmitt 
)
```

Configure the schmitt trigger for a pin.

**Parameters**

- `num` The GPIO number
- `schmitt` The Schmitt trigger configuration

**Returns**

On success a 0, otherwise -1

2.4.4.6 gpio_slew

```c
int8_t gpio_slew ( uint8_t num, 
    pad_slew_t slew 
)
```

Configure the slew rate for a pin.

**Parameters**

- `num` The GPIO number
- `slew` The slew rate of the pin

**Returns**

On success a 0, otherwise -1

2.4.4.7 gpio_read

```c
int8_t gpio_read ( uint8_t num )
```

Read a value from a GPIO pin.

**Parameters**

- `num` The GPIO number

**Returns**

The value of the pin (1 = high, 0 = low), otherwise -1
2.4.4.8  gpio_write

    int8_t gpio_write ( uint8_t num,
                        uint8_t val
                      )

Write a value to a GPIO pin.

Parameters

    num The GPIO number
    val  The value to write

Returns

    On success a 0, otherwise -1

2.4.4.9  gpio_toggle

    int8_t gpio_toggle ( uint8_t num )

Toggle the value of a GPIO pin.

Parameters

    num The GPIO number

Returns

    On success a 0, otherwise -1

2.4.4.10 gpio_interrupt_enable

    int8_t gpio_interrupt_enable ( uint8_t num,
                                   gpio_int_edge_t edge
                                  )

Enable an interrupt on a GPIO pin.

Parameters

    num The GPIO number
    edge The edge at which to trigger on

Returns

    On success a 0, otherwise -1

2.4.4.11 gpio_interrupt_disable

    int8_t gpio_interrupt_disable ( uint8_t num )

Disable an interrupt on a GPIO pin.

Parameters

    num The GPIO number

Returns

    On success a 0, otherwise -1
2.4.4.12 gpio_is_interrupted

```c
int8_t gpio_is_interrupted ( uint8_t num )
```

Check if an interrupt has happened on a GPIO pin.

**Parameters**

- `num` The GPIO number

**Returns**

On no interrupt 0, on an interrupt 1, otherwise -1
2.5 Assembler Definitions

The file `ft900_asm.h` contains the definitions for assembler instructions used in the `libft900.a` and `libft930.a` libraries.

2.5.1 API Cross Reference

No additional definitions are required.

2.5.2 Macro Documentation

2.5.2.1 `asm_noop`

```c
#define asm_noop()
```

A No Operation Instruction.

2.5.2.2 `asm_memcpy8`

```c
#define asm_memcpy8(src, dst, size)
```

8-bitwise memory copy.

**Parameters**

- **src** A pointer to the source data.
- **dst** A pointer to the destination data.
- **size** The size of the data to copy.

2.5.2.3 `asm_memcpy16`

```c
#define asm_memcpy16(src, dst, size)
```

16-bitwise memory copy.

**Parameters**

- **src** A pointer to the source data.
- **dst** A pointer to the destination data.
- **size** The size of the data to copy.

2.5.2.4 `asm_memcpy32`

```c
#define asm_memcpy32(src, dst, size)
```

32-bitwise memory copy.

**Parameters**

- **src** A pointer to the source data.
- **dst** A pointer to the destination data.
- **size** The size of the data to copy.
2.5.2.5 asm_memset8

#define asm_memset8(val, dst, size)

8-bitwise memory set.

Parameters

val The value to set the memory to.
dst A pointer to the destination data.
size The size of the data to copy.

2.5.2.6 asm_memset16

#define asm_memset16(val, dst, size)

16-bitwise memory set.

Parameters

val The value to set the memory to.
dst A pointer to the destination data.
size The size of the data to copy.

2.5.2.7 asm_memset32

#define asm_memset32(val, dst, size)

32-bitwise memory set.

Parameters

val The value to set the memory to.
dst A pointer to the destination data.
size The size of the data to copy.

2.5.2.8 asm_strcpy

#define asm_strcpy(src, dst)

String copy.

Parameters

src A pointer to the source string.
dst A pointer to the destination string.

2.5.2.9 asm_streamin8

#define asm_streamin8 (src, dst, size)

8-bitwise memory stream from FIFO to memory.

Parameters

src A pointer to the source registers.
dst A pointer to the destination data.
2.5.2.10 `asm_streamin16`

```c
#define asm_streamin16(src, dst, size)
```

16-bitwise memory stream from FIFO to memory.

**Parameters**

- `src` A pointer to the source registers.
- `dst` A pointer to the destination data.
- `size` The size of the data to copy.

2.5.2.11 `asm_streamin32`

```c
#define asm_streamin(src, dst, size)
```

32-bitwise memory stream from FIFO to memory.

**Parameters**

- `src` A pointer to the source registers.
- `dst` A pointer to the destination data.
- `size` The size of the data to copy.

2.5.2.12 `asm_streamout8`

```c
#define asm_streamout8(src, dst, size)
```

8-bitwise memory stream from memory to FIFO.

**Parameters**

- `src` A pointer to the source data.
- `dst` A pointer to the destination registers.
- `size` The size of the data to copy.

2.5.2.13 `asm_streamout16`

```c
#define asm_streamin16(src, dst, size)
```

16-bitwise memory stream from memory to FIFO.

**Parameters**

- `src` A pointer to the source data.
- `dst` A pointer to the destination registers.
- `size` The size of the data to copy.

2.5.2.14 `asm_streamout32`

```c
#define asm_streamout(src, dst, size)
```

32-bitwise memory stream from memory to FIFO.
Parameters

src  A pointer to the source data.

dst  A pointer to the destination registers.

size The size of the data to copy.

2.5.2.15 asm_setbit

#define asm_setbit(val, bit)
Set a bit in a 32 bit value.

Parameters

val  The value to use.

bit  The bit position to set.

2.5.2.16 asm_clrbit

#define asm_clrbit(val, bit)
Set a bit in a 32 bit value.

Parameters

val  The value to use.

bit  The bit position to clear.

2.5.2.17 asm_flip32

#define asm_flip32(src, dst, val)
Flip bit regions.

Parameters

src  A pointer to the source data.

dst  A pointer to the destination data.

val  The region of bits to flip.

- If bit 0 is set, then every alternate bit is exchanged.
- If bit 1 is set, then every alternate 2-bit group is exchanged.
- If bit 2 is set, then every alternate 4-bit group is exchanged.
- If bit 3 is set, then every alternate 8-bit group is exchanged.
- If bit 4 is set, then the two 16-bit groups are exchanged.

2.5.2.18 asm_reverse_endianness

#define asm_reverse_endianness(val)
Reverse the endianness of a value.

Parameters

val  The value to use.
2.5.2.19 `asm_reverse_bits`

```c
#define asm_reverse_bits (val)
```

Reverse the bits of a value.

**Parameters**

- `val` The value to use.

2.5.2.20 `asm_rotate32`

```c
#define asm_rotate32 (val, num)
```

Rotate bits left or right.

**Parameters**

- `val` The value to use.
- `num` The number and direction to rotate in (negative numbers rotate left).

2.5.3 Function Documentation

2.5.3.1 `asm_strncmp`

```c
static inline int32_t asm_strncmp(const char *src1, const char *src2)
```

String compare.

**Parameters**

- `src1` A pointer to the first source string.
- `src2` A pointer to the second source string.

**Returns**

The difference between the two strings.

2.5.3.2 `asm_strlen`

```c
static inline int32_t asm_strlen(const char *src)
```

String length.

**Parameters**

- `src` A pointer to the source string.

**Returns**

The length of the string.
2.6 Watchdog Timer

The file `ft900_wdt.h` contains the definitions for the watchdog timer functions in the libft900.a and libft930.a libraries.

2.6.1 API Cross Reference

Additional definitions are taken from:

`ft900_registers.h` – FT90x and FT93x register definitions

2.6.2 Enumeration Type Documentation

2.6.2.1 `wdt_counter_t`

```c
enum wdt_counter_t

Watchdog Timeouts.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Timeout (nsec @ 100 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdt_counter_1_clocks</td>
<td>10 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_2_clocks</td>
<td>20 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_4_clocks</td>
<td>40 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_8_clocks</td>
<td>80 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_16_clocks</td>
<td>160 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_32_clocks</td>
<td>320 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_64_clocks</td>
<td>640 nsec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_128_clocks</td>
<td>1.28 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_256_clocks</td>
<td>2.56 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_512_clocks</td>
<td>5.12 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_1K_clocks</td>
<td>10.24 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_2K_clocks</td>
<td>20.48 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_4K_clocks</td>
<td>40.96 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_8K_clocks</td>
<td>81.92 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_16K_clocks</td>
<td>163.84 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_32K_clocks</td>
<td>327.68 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_64K_clocks</td>
<td>655.35 usec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_128K_clocks</td>
<td>~1.31 msec @ 100 MHz</td>
</tr>
<tr>
<td>wdt_counter_256K_clocks</td>
<td>~2.62 msec @ 100 MHz</td>
</tr>
<tr>
<td>Function</td>
<td>Time (Msec) @ 100 MHz</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>wdt_counter_512K</td>
<td>~5.24</td>
</tr>
<tr>
<td>wdt_counter_1M</td>
<td>~10.49</td>
</tr>
<tr>
<td>wdt_counter_2M</td>
<td>~20.97</td>
</tr>
<tr>
<td>wdt_counter_4M</td>
<td>~41.94</td>
</tr>
<tr>
<td>wdt_counter_8M</td>
<td>~83.89</td>
</tr>
<tr>
<td>wdt_counter_16M</td>
<td>~167.77</td>
</tr>
<tr>
<td>wdt_counter_32M</td>
<td>~335.54</td>
</tr>
<tr>
<td>wdt_counter_64M</td>
<td>~671.09</td>
</tr>
<tr>
<td>wdt_counter_128M</td>
<td>~1.34</td>
</tr>
<tr>
<td>wdt_counter_256M</td>
<td>~2.68</td>
</tr>
<tr>
<td>wdt_counter_512M</td>
<td>~5.37</td>
</tr>
<tr>
<td>wdt_counter_1G</td>
<td>~10.74</td>
</tr>
<tr>
<td>wdt_counter_2G</td>
<td>~21.47</td>
</tr>
</tbody>
</table>

### 2.6.3 Function Documentation

#### 2.6.3.1 wdt_init

```c
int8_t wdt_init( wdt_counter_t timeout )
```

Initialise and start the Watchdog timer.

**Parameters**

- `timeout`: The timeout value of the Watchdog

**Returns**

- 0 on success, -1 otherwise

#### 2.6.3.2 wdt_kick

```c
int8_t wdt_kick( void )
```

Reset a running Watchdog Timer.

**Returns**

- 0 on success, -1 otherwise
2.7 Timers

The file `ft900_timers.h` contains the definitions for the timer management functions in the `libft900.a` and `libft930.a` libraries.

2.7.1 API Cross Reference

Additional definitions are taken from:

`ft900_registers.h` – FT90x and FT93x register definitions

2.7.2 Enumeration Type Documentation

2.7.2.1 `timer_select_t`

```
enum timer_select_t
{
    timer_select_a,  // Timer A
    timer_select_b,  // Timer B
    timer_select_c,  // Timer C
    timer_select_d,  // Timer D
};
```

2.7.2.2 `timer_direction_t`

```
enum timer_direction_t
{
    timer_direction_up,  // Count up
    timer_direction_down, // Count down
};
```

2.7.2.3 `timer_mode_t`

```
enum timer_mode_t
{
    timer_mode_continuous,  // Count continuous
    timer_mode_oneshot,    // Count one shot
};
```

2.7.2.4 `timer_prescaler_select_t`

```
enum timer_prescaler_select_t
{
};
```
2.7.3 Function Documentation

2.7.3.1 timer_init

```c
int8_t timer_init ( timer_select_t timer,
                 uint16_t initial,
                 timer_direction_t dir,
                 timer_prescaler_select_t prescaler,
                 timer_mode_t mode
)
```

Initialise a timer.

**Parameters**

- **timer** The timer to set up
- **initial** The initial value for the timer
- **dir** The direction that the timer should count in
- **prescaler** Whether or not this timer should use the prescaler
- **mode** If the timer should be continuously counting or a one shot

**Returns**

On success a 0, otherwise -1

2.7.3.2 timer_start

```c
int8_t timer_start ( timer_select_t timer )
```

Start a timer.

**Parameters**

- **timer** The timer to start

**Returns**

On success a 0, otherwise -1
2.7.3.3 timer_stop

    int8_t timer_stop ( timer_select_t timer )

Stop a timer.

Parameters
    timer The timer to stop

Returns

2.7.3.4 On success a 0, otherwise -1 timer_read

    int8_t timer_read ( timer_select_t timer,
                        uint16_t * value )

Read the value of a timer.

Parameters
    timer The timer to read from
    value A pointer to store the value

Returns
    On success a 0, otherwise -1

2.7.3.5 timer_prescaler

    int8_t timer_prescaler ( uint16_t prescaler ) [FT90X Revision B]
    int8_t timer_prescaler ( timer_select_t timer, uint16_t prescaler ) [FT93x and FT90x Revision C]

Set up the prescaler.

Parameters
    prescaler The clock prescaler to apply to the timer
    timer The timer to use [Only for FT93X]

Returns
    On success a 0, otherwise -1

Warning
    This can only be used before starting timers

Note:
    FT93X and FT90X series Revision C devices have separate prescalers for each timer, while on FT90X revision B, there is one common prescaler for all timers.

2.7.3.6 timer_disable_interrupt

    int8_t timer_disable_interrupt ( timer_select_t )

Disable the interrupt for a timer.
Parameters

- **timer** The timer to disable the interrupt for

Returns

On success a 0, otherwise -1

### 2.7.3.7 timer_enable_interrupt

```c
int8_t timer_enable_interrupt ( timer_select_t timer )
```

Enable the interrupt for a timer.

**Parameters**

- **timer** The timer to enable the interrupt for

Returns

On success a 0, otherwise -1

### 2.7.3.8 timer_is_interrupted

```c
int8_t timer_is_interrupted ( timer_select_t timer )
```

Check if a timer has been interrupted.

**Parameters**

- **timer** The timer to check

**Warning**

This function clears the current interrupt status bit

Returns

1 for if a timer is interrupted, 0 if the timer is not interrupted, -1 otherwise
2.8 Analogue to Digital Converter

The file `ft900_adc.h` contains the definitions for the analogue to digital conversion functions in the `libft900.a` and `libft930.a` libraries.

### 2.8.1 API Cross Reference

It utilises the following library APIs:

- `ft900_asm.h` – FT90X and FT93X assembler definitions
- Additional definitions are taken from:
  - `ft900_registers.h` – FT90X and FT93X register definitions

### 2.8.2 Enumeration Type Documentation

#### 2.8.2.1 `adc_resolution_t`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_10bit</td>
<td>10-bit ADC resolution</td>
</tr>
<tr>
<td>adc_8bit</td>
<td>8-bit ADC resolution</td>
</tr>
</tbody>
</table>

#### 2.8.2.2 `adc_clock_t`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_12_5_MHz</td>
<td>ADC clock is 12.5 MHz</td>
</tr>
<tr>
<td>adc_6_25_MHz</td>
<td>ADC clock is 6.25 MHz</td>
</tr>
</tbody>
</table>

#### 2.8.2.3 `adc_mode_t`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_mode_single</td>
<td>One analogue reading will be taken and then the ADC stopped</td>
</tr>
<tr>
<td>adc_mode_continuous</td>
<td>The ADC will continuously acquire analogue readings</td>
</tr>
</tbody>
</table>
2.8.3 Function Documentation

2.8.3.1 adc_start

```c
int8_t adc_start ( uint8_t channel )
```

Start the ADC capturing.

**Parameters**

- **channel** The channel to select

**Returns**

0 on success, -1 otherwise

2.8.3.2 adc_stop

```c
int8_t adc_stop ( void )
```

Stop the ADC from capturing.

**Returns**

0 on success, -1 otherwise

2.8.3.3 adc_mode

```c
int8_t adc_mode ( adc_mode_t mode )
```

Choose the mode that the ADC will run in.

**Parameters**

- **mode** The mode (single or continuous)

**Returns**

0 on success, -1 otherwise

2.8.3.4 adc_select_resolution

```c
int8_t adc_select_resolution(adc_resolution_t resolution)
```

Choose the ADC to be 10-bit or 8-bit resolution. The ADC resolution is configurable ONLY in FT90x Revision C.

**Parameters**

- **resolution** The resolution (10 or 8 bit)

**Returns**

0 on success, -1 otherwise (in case of FT930 or FT90x Revision B)

2.8.3.5 adc_select_frequency

```c
int8_t adc_select_frequency(adc_clock_t frequency)
```

Choose the ADC clock for sampling to be 12.5MHz or half of this rate at 6.25MHz. The ADC clock is configurable ONLY in FT90x Revision C.

**Parameters**
frequency The clock frequency (12.5MHz or 6.25MHz)

Returns
0 on success, -1 otherwise (in case of FT930 or FT90x Revision B)

2.8.3.6  adc_read

int8_t adc_read ( uint16_t * val )

Get the next sample from the ADC.

Parameters
val A pointer to store the sample

Returns
The number of samples read, -1 otherwise

2.8.3.7  adc_readn

int adc_readn ( uint16_t * val, size_t len )

Get a collection of samples from the ADC and store it in the array passed as parameter.

Parameters
val An array pointer to store the samples
len The number of 10-bit word samples to read from the FIFO

Warning
This function will only work when the ADC is in continuous mode

Returns
The number of word samples read, -1 otherwise

2.8.3.8  adc_available

uint8_t adc_available ( void )

Get the number of ADC samples available.

Returns
The number of ADC samples

Note: In case of FT90x series Revision B, this function should be called in interrupt context for a reliable count of the ADC samples available. For FT90X Revision C or FT93X, this API can be used in interrupt or polling method of servicing the ADC application.

2.8.3.9  adc_disable_interrupt

int8_t adc_disable_interrupt ( void )

Disable the Interrupt on the ADC.
Returns
0 on success, -1 otherwise

2.8.3.10 adc_enable_interrupt

int8_t adc_enable_interrupt ( void )

Enable the Interrupt on the ADC.

Returns
0 on success, -1 otherwise

2.8.3.11 adc_is_interrupted

int8_t adc_is_interrupted ( void )

Check that the ADC has been interrupted.

Warning
This function will clear the current interrupt bit when called

Returns
1 interrupted, 0 not interrupted
2.9 Digital to Analogue Converter

The file `ft900_dac.h` contains the definitions for the digital to analogue conversion functions in the `libft900.a` and `libft930.a` libraries.

### 2.9.1 API Cross Reference

It utilises the following library APIs:
- `ft900_asm.h` – FT90X and FT93X assembler definitions
- Additional definitions are taken from:
  - `ft900_registers.h` – FT90X and FT93X register definitions

### 2.9.2 Enumeration Type Documentation

#### 2.9.2.1 `dac_mode_t`

```c
enum dac_mode_t

DAC run mode.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dac_mode_single</code></td>
<td>Run the DAC in Single Shot mode</td>
</tr>
<tr>
<td><code>dac_mode_continuous</code></td>
<td>Run the DAC in Continuous mode</td>
</tr>
</tbody>
</table>

### 2.9.3 Function Documentation

#### 2.9.3.1 `dac_start`

```c
int8_t dac_start ( uint8_t num )
```

Start the DAC.

**Parameters**

- `num` The DAC to use

**Returns**

0 on success, -1 otherwise

#### 2.9.3.2 `dac_stop`

```c
int8_t dac_stop ( uint8_t num )
```

Stop the DAC.

**Parameters**

- `num` The DAC to use

**Returns**

0 on success, -1 otherwise
2.9.3.3 *dac_mode*

```c
int8_t dac_mode( uint8_t num, 
                 dac_mode_t mode )
```

Select the mode of the DAC.

**Parameters**

- `num` The DAC to use
- `mode` The mode the DAC should be in (single or continuous)

**Returns**

0 on success, -1 otherwise

2.9.3.4 *dac_divider*

```c
int8_t dac_divider( uint8_t div )
```

Select the divider for the DAC conversion rate.

\[ f_{DAC} = \frac{f_{peripheral}}{\text{div}+1} \]

**Parameters**

- `div` The divider

**Warning**

The maximum conversion rate is 1 MHz

**Returns**

0 on success, -1 otherwise

2.9.3.5 *dac_write*

```c
int8_t dac_write( uint8_t num, 
                  uint16_t data )
```

Write a value to the DAC.

This function will automatically update the DAC when it is in single mode.

**Parameters**

- `num` The DAC to use
- `data` The sample to write

**Returns**

The number of bytes written, -1 otherwise
2.9.3.6  `dac_writen`

```c
int dac_writen ( uint8_t num,
    uint16_t * data,
    size_t len )
```

Write a series of values to the DAC.

**Parameters**

- `num` The DAC to use
- `data` The array of samples to write
- `len` The number of samples to write

**Warning**

This function will only work when continuous mode is selected

**Returns**

The number of samples written, -1 otherwise

2.9.3.7  `dac_available`

```c
uint8_t dac_available ( uint8_t num )
```

See how many samples are still being converted.

**Parameters**

- `num` The DAC to use

**Returns**

The number of samples still to be converted, or 0 otherwise

2.9.3.8  `dac_disable_interrupt`

```c
int8_t dac_disable_interrupt ( uint8_t num )
```

Disable the interrupt on a DAC.

**Parameters**

- `num` The DAC to use

**Returns**

0 on success, -1 otherwise
2.9.3.9  *dac_enable_interrupt*

```c
int8_t dac_enable_interrupt ( uint8_t num )
```

Enable the interrupt on a DAC.
Enable the DAC module to generate an interrupt. The DAC module will generate an interrupt after 64 samples have been generated on the DAC.

**Parameters**

- `num` The DAC to use

**Returns**

0 on success, -1 otherwise

2.9.3.10 *dac_is_interrupted*

```c
int8_t dac_is_interrupted ( uint8_t num )
```

Check if the DAC has fired an interrupt.

**Parameters**

- `num` The DAC to use

**Warning**

This function clears the current interrupt status bit

**Returns**

1 when interrupted, 0 when not interrupted, -1 otherwise
2.10 Ethernet driver

The file `ft900_eth.h` contains the definitions for the Ethernet management and control functions in the libft900.a library.

### 2.10.1 API Cross Reference

It utilises the following library APIs:

- `ft900_sys.h` – Chip Management
- `ft900_gpio.h` – General Purpose I/O and Pad Control

Additional definitions are taken from:

- `ft900_registers.h` – FT90X register definitions

### 2.10.2 Enumeration Type Documentation

#### 2.10.2.1 `ethernet_led_mode_t`

```c
eenum ethernetLedMode_t
```

Ethernet LED pin mode.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ethernetLedMode_link</code></td>
<td>Link active</td>
</tr>
<tr>
<td><code>ethernetLedMode_tx</code></td>
<td>Transmit</td>
</tr>
<tr>
<td><code>ethernetLedMode_rx</code></td>
<td>Receive</td>
</tr>
<tr>
<td><code>ethernetLedMode_col</code></td>
<td>Collision</td>
</tr>
<tr>
<td><code>ethernetLedMode_fdx</code></td>
<td>Full duplex</td>
</tr>
<tr>
<td><code>ethernetLedMode_spd</code></td>
<td>Speed 10/100</td>
</tr>
</tbody>
</table>

### 2.10.3 Function Documentation

#### 2.10.3.1 `ethernet_init`

```c
void ethernet_init ( const uint8_t * mac )
```

Initialise the ethernet hardware.

**Parameters**

- `mac` pointer to a six byte array containing MAC

#### 2.10.3.2 `ethernet_tx_enable`

```c
void ethernet_tx_enable ( int flag )
```

Enable or disable the Ethernet transmitter.

**Parameters**

- `flag` 0 - disable transmitter, 1 - enable transmitter
2.10.3.3 ethernet_rx_enable

`void ethernet_rx_enable ( int flag )`

Enable or disable the Ethernet receiver.

**Parameters**

- **flag** 0 - disable receiver, 1 - enable receiver

2.10.3.4 ethernet_led_mode

`void ethernet_led_mode ( uint8_t led, ethernet_led_mode_t mode )`

Set the mode of the led.

**Parameters**

- **led** The number of the led (0 or 1)
- **mode** The mode which the led will be in

2.10.3.5 ethernet_set_mac

`void ethernet_set_mac ( const uint8_t * mac )`

Set the Ethernet MAC.

**Parameters**

- **mac** A pointer to a six byte array containing MAC

2.10.3.6 ethernet_set_promiscuous

`void ethernet_set_promiscuous ( int flag )`

Set the Ethernet peripheral in promiscuous mode.

**Parameters**

- **flag** 0 - reset promiscuous mode, 1 - set promiscuous mode

2.10.3.7 ethernet_read

`int ethernet_read ( size_t * blen, uint8_t * buf )`

Poll the ethernet peripheral for the reception of a single packet.

**Parameters**

- **buf** Pointer to buffer to store the received packet.
- **blen** Size of reception buffer

**Returns**

- 1 packet received, 0 no packet received
2.10.3.8 ethernet_mii_read

uint16_t ethernet_mii_read ( uint8_t reg )

Read the content of an MII register.

Parameters

reg register to read

Returns

The content of requested register

2.10.3.9 ethernet_write

int ethernet_write ( uint8_t * buf,
    size_t blen
)

Outputs a packet on the ethernet interface.

The buffer must be in the following format: To send (n) bytes

<table>
<thead>
<tr>
<th>Offset</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf[0]</td>
<td>lsb of payload length (n &amp; 0xff)</td>
</tr>
<tr>
<td>buf[1]</td>
<td>msb of payload length (n &gt;&gt; 16)</td>
</tr>
<tr>
<td>buf[2]</td>
<td>destination MAC[0]</td>
</tr>
<tr>
<td>buf[8]</td>
<td>source MAC[0]</td>
</tr>
<tr>
<td>buf[14]</td>
<td>packet type</td>
</tr>
<tr>
<td>buf[15]</td>
<td>packet type</td>
</tr>
<tr>
<td>Offset</td>
<td>Use</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>buf[16]</td>
<td>payload</td>
</tr>
<tr>
<td>buf[,]</td>
<td>...</td>
</tr>
<tr>
<td>buf[n+16]</td>
<td>end of payload (n bytes).</td>
</tr>
</tbody>
</table>

**Parameters**

- **buf**  Pointer to packet to send
- **blen** Length of packet to send (in bytes)

### 2.10.3.10  ethernet_mii_write

```c
int ethernet_mii_write ( uint8_t reg,
                         uint16_t v )
```

Write a value to an MII register.

**Parameters**

- **reg**   Register to write to
- **v**    Value to write to requested register

**Returns**

0 on success, -1 on error

### 2.10.3.11  ethernet_is_link_up

```c
int ethernet_is_link_up ( void )
```

Return the Ethernet link status.

**Returns**

1 - link up, 0 - link is not up.

### 2.10.3.12  ethernet_enable_interrupt

```c
void ethernet_enable_interrupt ( uint8_t mask )
```

Enable the Ethernet peripheral to fire interrupts.

**Parameters**

- **mask**
2.11 UART

The file `ft900_simple_uart.h` contains the definitions for the UART management and control functions in the `libft900.a` library.

2.11.1 API Cross Reference

It utilises the following library APIs:

- `ft900_asm.h` – FT90X and FT93X assembler definitions

Additional definitions are taken from:

- `ft900_registers.h` – FT90X and FT93X register definitions

2.11.2 Macro Definition Documentation

```c
#define UART_DIVIDER_1000000_BAUD  (25)  // Predefined divider for 1000000 baud.
#define UART_DIVIDER_110_BAUD      (56818)  // Predefined divider for 110 baud.
#define UART_DIVIDER_115200_BAUD   (217)   // Predefined divider for 115200 baud.
#define UART_DIVIDER_1200_BAUD     (20833)  // Predefined divider for 1200 baud.
#define UART_DIVIDER_150_BAUD      (55555)  // Predefined divider for 150 baud.
#define UART_DIVIDER_19200_BAUD    (1302)   // Predefined divider for 19200 baud.
#define UART_DIVIDER_230400_BAUD   (109)    // Predefined divider for 230400 baud.
#define UART_DIVIDER_2400_BAUD     (10417)  // Predefined divider for 2400 baud.
#define UART_DIVIDER_300_BAUD      (27778)  // Predefined divider for 300 baud.
#define UART_DIVIDER_31250_BAUD    (800)    // Predefined divider for 31250 baud.
#define UART_DIVIDER_38400_BAUD    (651)    // Predefined divider for 38400 baud.
#define UART_DIVIDER_460800_BAUD   (54)     // Predefined divider for 460800 baud.
#define UART_DIVIDER_4800_BAUD     (5208)   // Predefined divider for 4800 baud.
```
# define UART_DIVIDER_57600_BAUD (434)
Predefined divider for 57600 baud.

# define UART_DIVIDER_921600_BAUD (27)
Predefined divider for 921600 baud.

# define UART_DIVIDER_9600_BAUD (2604)
Predefined divider for 9600 baud.

### 2.11.3 Enumeration Type Documentation

#### 2.11.3.1 uart_data_bits_t

`enum uart_data_bits_t`

 UART Data bits.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_data_bits_5</td>
<td>5 data bits</td>
</tr>
<tr>
<td>uart_data_bits_6</td>
<td>6 data bits</td>
</tr>
<tr>
<td>uart_data_bits_7</td>
<td>7 data bits</td>
</tr>
<tr>
<td>uart_data_bits_8</td>
<td>8 data bits</td>
</tr>
</tbody>
</table>

#### 2.11.3.2 uart_interrupt_t

`enum uart_interrupt_t`

 UART Interrupts.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_interrupt_none</td>
<td>No Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_tx</td>
<td>Transmit Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_rx</td>
<td>Receive Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_dcd_ri_dsr_cts</td>
<td>DCD/RI/DSR/CTS Change Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_xon_xoff</td>
<td>In-band flow control Interrupt (in 16950 mode only)</td>
</tr>
<tr>
<td>uart_interrupt_rts_cts</td>
<td>Out-of-band flow control Interrupt (in 16950 mode only)</td>
</tr>
</tbody>
</table>

#### 2.11.3.3 uart_parity_t

`enum uart_parity_t`

 UART Parity bits.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
</table>

Product Page  
Document Feedback

Copyright © Bridgetek Pte Ltd
### uart_parity_t

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_parity_none</td>
<td>No parity</td>
</tr>
<tr>
<td>uart_parity_odd</td>
<td>Odd parity</td>
</tr>
<tr>
<td>uart_parity_even</td>
<td>Even parity</td>
</tr>
</tbody>
</table>

#### 2.11.3.4 uart_stop_bits_t

**Enum uart_stop_bits_t**

UART Stop bits.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_stop_bits_1</td>
<td>1 stop bit</td>
</tr>
<tr>
<td>uart_stop_bits_1_5</td>
<td>1.5 stop bit</td>
</tr>
<tr>
<td>uart_stop_bits_2</td>
<td>2 stop bit</td>
</tr>
</tbody>
</table>

#### 2.11.3.5 uart_flow_t

**Enum uart_flow_t**

UART Flow control.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_flow_none</td>
<td>No flow control</td>
</tr>
<tr>
<td>uart_flow_rts_cts</td>
<td>RTS/CTS flow control</td>
</tr>
<tr>
<td>uart_flow_dtr_dsr</td>
<td>DTR/DSR flow control</td>
</tr>
<tr>
<td>uart_flow_xon_xoff</td>
<td>XON/XOFF flow control</td>
</tr>
</tbody>
</table>

#### 2.11.3.6 uart_mode_t

**Enum uart_mode_t**

UART Mode control.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_mode_16450</td>
<td>16450 mode No FIFO enabled</td>
</tr>
<tr>
<td>uart_mode_16550</td>
<td>16550 mode 16 byte FIFO enabled</td>
</tr>
<tr>
<td>uart_mode_16650</td>
<td>16650 mode 128 byte FIFO enabled, autoRTS/CTS, XON/XOFF</td>
</tr>
<tr>
<td>uart_mode_16750</td>
<td>16750 mode 128 byte FIFO enabled, autoRTS/CTS</td>
</tr>
<tr>
<td>uart_mode_16950</td>
<td>16950 mode 128 byte FIFO enabled, autoRTS/CTS, autoDTR/DSR, XON/XOFF, RS485</td>
</tr>
</tbody>
</table>
2.11.3.7 uart_interrupt_t

```c
enum uart_interrupt_t
UART interrupts.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_interrupt_none</td>
<td>No Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_tx</td>
<td>Transmit Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_rx</td>
<td>Receive Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_9th_bit</td>
<td>9th bit set interrupt in 9-bit mode and also Receiver error interrupt</td>
</tr>
<tr>
<td>uart_interrupt_rx_time_out</td>
<td>Receive TimeOut Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_dcd_ri_dsr_cts</td>
<td>DCD/RI/DSR/CTS Change Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_xon_xoff</td>
<td>In-band flow control Interrupt (in 16950 mode only) along with Special character Interrupt for 9-bit data mode and Special character Interrupt</td>
</tr>
<tr>
<td>uart_interrupt_rts_cts</td>
<td>Out-of-band flow control Interrupt (in 16950 mode only)</td>
</tr>
</tbody>
</table>

2.11.4 Function Documentation

2.11.4.1 uart_open

```c
int8_t uart_open ( ft900_uart_regs_t * dev,
                   uint8_t prescaler,
                   uint32_t divisor,
                   uart_data_bits_t databits,
                   uart_parity_t parity,
                   uart_stop_bits_t stop
)
```

Open a UART for communication.

**Parameters**

- `dev` The device to use
- `prescaler` The value of the prescaler
- `divisor` The value of the divisor
- `databits` The number of data bits
- `parity` The parity scheme
stop

The number of stop bits

Warning
1.5 stop bits is only available in 5 bit mode

Returns
0 on success, -1 otherwise (invalid device)

2.11.4.2 uart_close

int8_t uart_close ( ft900_uart_regs_t * dev )

Close a UART from communication.

Parameters
  dev The device to use

Returns
0 on success, -1 otherwise (invalid device)

2.11.4.3 uart_calculate_baud

int32_t uart_calculate_baud ( uint32_t target_baud,
                      uint8_t samples,
                      uint32_t f_perif,
                      uint16_t * divisor,
                      uint8_t * prescaler )

Calculate the prescaler and divisor from a baudrate.

Parameters
  target_baud The baud rate to use
  samples The number of samples the UART will take for a bit, the default for this is 4
  f_perif Peripheral frequency, the default for this is 100,000,000
  divisor A pointer to store the divisor
  prescaler A pointer to store the prescaler, if this is NULL the prescaler will be set to 1

For Nsamples = 4 the following baud rates can be obtained from these Divisors and Prescalers:

Table 5- UART Baudrate table

<table>
<thead>
<tr>
<th>Desired baud</th>
<th>Prescaler</th>
<th>Divisor</th>
<th>Actual baud</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>4</td>
<td>56818</td>
<td>110.00035</td>
<td>~0.000%</td>
</tr>
<tr>
<td>150</td>
<td>3</td>
<td>55555</td>
<td>150.00150</td>
<td>+0.001%</td>
</tr>
<tr>
<td>300</td>
<td>3</td>
<td>27778</td>
<td>299.99760</td>
<td>~0.000%</td>
</tr>
</tbody>
</table>
Returns
The absolute error from the target baud rate

2.11.4.4 uart_puts

size_t uart_puts ( ft900_uart_regs_t * dev,
                    char * str )

Write a string to the serial port.
Parameters
    dev The device to use
    str The null-terminated string to write

Returns
The number of bytes written or -1 otherwise

2.11.4.5 uart_read

size_t uart_read ( ft900_uart_regs_t * dev,
                   uint8_t * buffer )

Read a data word from a UART.
Parameters
    dev The device to use
buffer A pointer to the data word to store into

**Returns**
The number of bytes read or -1 otherwise (invalid device)

### 2.11.4.6 uart_readn

```c
size_t uart_readn ( ft900_uart_regs_t * dev,
                  uint8_t * buffer,
                  size_t len
)
```

Read a series of data words from a UART.

**Parameters**
- **dev**  The device to use
- **buffer** A pointer to the array of data words to store into
- **len**  The number of data words to read

**Returns**
The number of bytes read or -1 otherwise (invalid device)

### 2.11.4.7 uart_write

```c
size_t uart_write ( ft900_uart_regs_t * dev,
                   uint8_t * buffer
)
```

Write a data word to a UART.

**Parameters**
- **dev**  The device to use
- **buffer** The data to send

**Returns**
The number of bytes written or -1 otherwise (invalid device)

### 2.11.4.8 uart_writen

```c
size_t uart_writen ( ft900_uart_regs_t * dev,
                     uint8_t * buffer,
                     size_t len
)
```

Write a series of data words to a UART.

**Parameters**
- **dev**  The device to use
buffer A pointer to the array to send
len The size of buffer

Returns
The number of bytes written or -1 otherwise (invalid device)

2.11.4.9 uart_is_interrupted

int8_t uart_is_interrupted ( ft900_uart_regs_t * dev,
    uart_interrupt_t interrupt )

Check if an interrupt has been triggered.

Parameters
    dev The device to use
    interrupt The interrupt to check

Warning
This function clears the current interrupt status bit

Returns
1 when interrupted, 0 when not interrupted, -1 otherwise (invalid device)

2.11.4.10 uart_enable_interrupt

int8_t uart_enable_interrupt ( ft900_uart_regs_t * dev,
    uart_interrupt_t interrupt )

Enable an interrupt on the UART.

Parameters
    dev The device to use
    interrupt The interrupt to enable

Returns
0 on success, -1 otherwise (invalid device)

2.11.4.11 uart_enable_interruptions_globally

int8_t uart_enable_interruptions_globally ( ft900_uart_regs_t * dev )

Enable a UART to interrupt.

Parameters
    dev The device to use

Returns
0 on success, -1 otherwise (invalid device)
2.11.4.12 \textit{uart_disable_interrupt}

\begin{verbatim}
int8_t uart_disable_interrupt ( ft900_uart_regs_t * dev, 
        uart_interrupt_t interrupt 
    )
\end{verbatim}

Disable an interrupt on the UART.

\textbf{Parameters}
- \texttt{dev} The device to use
- \texttt{interrupt} The interrupt to disable

\textbf{Returns}
- 0 on success, -1 otherwise (invalid device)

2.11.4.13 \textit{uart_disable_interrupts_globally}

\begin{verbatim}
int8_t uart_disable_interrupts_globally ( ft900_uart_regs_t * dev )
\end{verbatim}

Disable a UART from interrupting.

\textbf{Parameters}
- \texttt{dev} The device to use

\textbf{Returns}
- 0 on success, -1 otherwise (invalid device)

2.11.4.14 \textit{uart_get_interrupt}

\begin{verbatim}
uint8_t uart_get_interrupt(ft900_uart_regs_t *dev)
\end{verbatim}

Return the currently indicated interrupt.

\textbf{Parameters}
- \texttt{dev} The device to use

\textbf{Returns}
- enum of possible interrupt levels.
  Cast to values defined in enum uartInterrupt_t or 0xff if invalid device.

2.11.4.15 \textit{uart_rts}

\begin{verbatim}
int8_t uart_rts(ft900_uart_regs_t *dev, int active)
\end{verbatim}

Enable or disable RTS signal.

\textbf{Parameters}
- \texttt{dev} The device to use
- \texttt{active} Non-zero to enable RTS (line high) zero to disable (line low)

\textbf{Returns}
- 0 on success or -1 otherwise(invalid device).

2.11.4.16 \textit{uart_dtr}

\begin{verbatim}
int8_t uart_dtr(ft900_uart_regs_t *dev, int active)
\end{verbatim}

Enable or disable DTR signal.
Parameters

- **dev** The device to use
- **active** Non-zero to enable DTR (line high) zero to disable (line low)

Returns

0 on success or -1 otherwise (invalid device).

2.11.4.17 **uart_cts**

```c
int8_t uart_cts(ft900_uart_regs_t *dev)
```

Test status of CTS signal.

Parameters

- **dev** The device to use

Returns

1 when CTS enabled, 0 when not enabled, -1 otherwise (invalid device)

2.11.4.18 **uart_dsr**

```c
int8_t uart_dsr(ft900_uart_regs_t *dev)
```

Test status of DSR signal.

Parameters

- **dev** The device to use

Returns

1 when DSR enabled, 0 when not enabled, -1 otherwise (invalid device)

2.11.4.19 **uart_ri**

```c
int8_t uart_ri(ft900_uart_regs_t *dev)
```

Test status of RI signal.

Parameters

- **dev** The device to use

Returns

1 when RI enabled, 0 when not enabled, -1 otherwise (invalid device)

2.11.4.20 **uart_dcd**

```c
int8_t uart_dcd(ft900_uart_regs_t *dev)
```

Test status of DCD signal.

Parameters

- **dev** The device to use

Returns

1 when DCD enabled, 0 when not enabled, -1 otherwise (invalid device)

2.11.4.21 **uart_mode**

```c
int8_t uart_mode(ft900_uart_regs_t *dev, uart_mode_t mode)
```

Set the mode of the UART. After the mode is selected all flow control and UART settings are reset. The uart_open function must be called again to re-initialise the UART.
## Parameters

- **dev** The device to use
- **mode** The mode to select

## Returns

0 if successful, -1 otherwise (invalid device)

### 2.11.4.22 `uart_set_trigger_level`

```c
int8_t uart_set_trigger_level(ft900_uart_regs_t *dev, uart_mode_t mode,
uart_fifo_trigger_level_t rxIntLevel, uart_fifo_trigger_level_t txIntLevel,
uint8_t FCH, uint8_t FCL)
```

Set TriggerLevel for various UART modes (550/650/750/950).

**Parameters**

- **dev** The device to use
- **mode** The UART mode
- **rxIntLevel** Receiver interrupt level
- **txIntLevel** Transmitter interrupt level
- **FCH** Flow Control High
- **FCL** Flow Control Low

**Returns**

0 if successful, -1 otherwise (invalid device or invalid parameter)

### 2.11.4.23 `uart_set_flow_control`

```c
int8_t uart_set_flow_control(ft900_uart_regs_t *dev, uart_flow_t mode,
uint8_t xOn, uint8_t xOff)
```

Set flow control.

**Parameters**

- **dev** The device to use
- **mode** The flow control mode to enable
- **xOn** xOn character (only used in XON-XOFF flow control, ignored in other cases)
- **xOff** xOff character (only used in XON-XOFF flow control, ignored in other cases)

**Returns**

0 if successful, -1 otherwise (invalid device)

### 2.11.4.24 `uart_get_rx_fifo_level`

```c
uint8_t uart_get_rx_fifo_level(ft900_uart_regs_t *dev)
```

Get receiver FIFO level.

**Parameters**

- **dev** The device to use

**Returns**
2.11.4.25  uart_get_tx_fifo_level

uint8_t uart_get_tx_fifo_level(ft900_uart_regs_t *dev)

Get transmitter FIFO level.

Parameters

   dev  The device to use

Returns

Number of bytes in transmitter FIFO

2.11.4.26  uart_flush_rx_fifo

uint8_t uart_flush_rx_fifo(ft900_uart_regs_t *dev)

Clears all bytes in the receiver FIFO and resets its counter logic to 0.

Note: The shift register is not cleared.

Parameters

   dev  The device to use

Returns

0 if successful, -1 otherwise (invalid device)

2.11.4.27  uart_flush_tx_fifo

uint8_t uart_flush_tx_fifo(ft900_uart_regs_t *dev)

Clears all bytes in the transmitter FIFO and resets its counter logic to 0.

Note: The shift register is not cleared.

Parameters

   dev  The device to use

Returns

0 if successful, -1 otherwise (invalid device)

2.11.4.28  uart_configure_9bit_address

int8_t uart_configure_9bit_address(ft900_uart_regs_t *dev, uint16_t address_1, uint16_t address_2, uint16_t address_3, uint16_t address_4)

Configure and enable 9-bit mode for sending and receiving 9-bit address and data. Upto four addresses can be configured for 9-bit mode address detection. Before configuring 9-bit address, UART has to be enabled for uart_mode_16950.

Note: If only 1 address has to be configured for address matching, then all four address parameters should be configured with the same address.

Parameters

   dev  The device to use

      address_1  One among four addresses for address detection

      address_2  One among four addresses for address detection

      address_3  One among four addresses for address detection

      address_4  One among four addresses for address detection

Returns

0 if successful, -1 otherwise (invalid device)
2.11.4.29  *uart_send_9bit_address*

```c
int8_t uart_send_9bit_address( ft900_uart_regs_t * dev, uint16_t  address )
```

Send 9-bit address from UART. Before sending 9-bit address, UART has to be enabled for 9-bit mode.

**Parameters**

- `dev` The device to use
- `address` The 9 bit address to send out

**Returns**

0 if successful, -1 otherwise (invalid device)

2.11.4.30  *uart_soft_reset*

```c
void uart_soft_reset(ft900_uart_regs_t *dev)
```

Issues UART Soft reset, which resets all UART registers to default value except clock select (CKS) and clock alteration (CKA) registers.

**Parameters**

- `dev` The device to use

**Returns**

None

### 2.12 I²C Master

The file `ft900_i2cm.h` contains the definitions for the I²C master bus functions in the `libft900.a` and `libft930.a` libraries

#### 2.12.1 API Cross Reference

It utilises the following library APIs:

- `ft900_asm.h` – FT90X and FT93X assembler definitions

Additional definitions are taken from:

- `ft900_registers.h` – FT90X and FT93X register definitions

#### 2.12.2 Enumeration Type Documentation

#### 2.12.2.1 I2CM_speed_mode

```c
enum I2CM_speed_mode
```

Available speed modes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Speed range</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2CM_NORMAL_SPEED</td>
<td>98Kbps to 100 kbps</td>
</tr>
<tr>
<td>I2CM_FAST_SPEED</td>
<td>130Kbps to 400 kbps</td>
</tr>
</tbody>
</table>
2.12.2.2 I2CM_clk_speeds

enum I2CM_clk_speeds

Some common I2C clock speeds.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2CM_100_Kbps</td>
<td>For 100 kbps</td>
</tr>
<tr>
<td>I2CM_400_Kbps</td>
<td>For 400 kbps</td>
</tr>
<tr>
<td>I2CM_1000_Kbps</td>
<td>For 1000 kbps</td>
</tr>
<tr>
<td>I2CM_3400_Kbps</td>
<td>For 3400 kbps</td>
</tr>
</tbody>
</table>

2.12.3 Function Documentation

2.12.3.1 i2cm_init

```c
void i2cm_init ( I2CM_speed_mode mode,
                uint32_t i2c_clk_speed )
```

Call once, to initialise the master and reset it to a known state.
It is not valid to call any other I2C Master functions before this one.

**Parameters**

- `[in] mode` I2C Master Clock mode
- `[in] i2c_clk_speed` The clock rate of the I2C bus. Enums of I2C_clk_speeds can be used.

2.12.3.2 i2cm_read

```c
int8_t i2cm_read ( const uint8_t addr,
                    const uint8_t cmd,
                    uint8_t * data,
                    const uint16_t number_to_read )
```

Read a specified number of bytes from an I2C device.
Automatically adopts burst mode if the slave supports it, and more than one byte is to be read.

**Parameters**

- `[in] addr` I2C address to write to.
- `[in] cmd` I2C command byte
[in] `data` Buffer containing bytes to write.

[in] `number_to_read` Number of bytes to read.

**Returns**

0 on success, -1 on an error

### 2.12.3.3 `i2cm_write`

```c
int8_t i2cm_write ( const uint8_t addr,  
                   const uint8_t cmd,  
                   const uint8_t * data,  
                   const uint16_t number_to_write )
```

Write a specified number of bytes to an I2C device. Automatically adopts burst mode if the slave supports it, and more than one byte is to be written.

**Parameters**

- [in] `addr` I2C address to write to.
- [in] `cmd` I2C command byte
- [in] `data` Buffer containing bytes to write.
- [in] `number_to_write` Number of bytes to read.

**Returns**

#### 2.12.3.40 on success, -1 on an error

### `i2cm_get_status`

```c
uint8_t i2cm_get_status ( void )
```

Determine I2C Master status.

#### Table 6- I2CM Status Mask

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask Name</th>
<th>Set when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (LSB)</td>
<td>MASK_I2CM_STATUS_I2C_BUSY</td>
<td>The Bus is currently busy transmitting/receiving</td>
</tr>
<tr>
<td>1</td>
<td>MASK_I2CM_STATUS_I2C_ERR</td>
<td>An error occurred (ADDR_ACK, DATA_ACK, ARB_LOST)</td>
</tr>
<tr>
<td>2</td>
<td>MASK_I2CM_STATUS_ADDR_ACK</td>
<td>Slave address was not acknowledged</td>
</tr>
<tr>
<td>3</td>
<td>MASK_I2CM_STATUS_DATA_ACK</td>
<td>Data was not acknowledged</td>
</tr>
<tr>
<td>4</td>
<td>MASK_I2CM_STATUS_ARB_LOST</td>
<td>Arbitration lost</td>
</tr>
<tr>
<td>5</td>
<td>MASK_I2CM_STATUS_I2C_IDLE</td>
<td>The I2C Controller is idle</td>
</tr>
<tr>
<td>6</td>
<td>MASK_I2CM_STATUS_BUS_BUSY</td>
<td>The I2C Bus is busy</td>
</tr>
<tr>
<td>7 (MSB)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Returns
0 on success, -1 on an error

2.12.3.5 i2cm_interrupt_disable

int8_t i2cm_interrupt_disable ( uint8_t mask )
Disable an interrupt.

See also
i2cm_interrupt_enable

Parameters
mask The bit pattern of interrupts to disable

Returns
0 on success, -1 on an error

2.12.3.6 i2cm_interrupt_enable

int8_t i2cm_interrupt_enable ( uint8_t mask )
Enable an interrupt.

Parameters
mask The bit pattern of interrupts to enable

Table 7- I2CM FIFO interrupt Mask

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask Name</th>
<th>Interrupts on...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (LSB)</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_TX_EMPTY</td>
<td>When the Transmit FIFO is empty</td>
</tr>
<tr>
<td>1</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_TX_HALF</td>
<td>When the Transmit FIFO is half empty</td>
</tr>
<tr>
<td>2</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_TX_FULL</td>
<td>When the Transmit FIFO is full</td>
</tr>
<tr>
<td>3</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_RX_EMPTY</td>
<td>When the Receive FIFO is empty</td>
</tr>
<tr>
<td>4</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_RX_HALF</td>
<td>When the Receive FIFO is half full</td>
</tr>
<tr>
<td>5</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_RX_FULL</td>
<td>When the Receive FIFO is full</td>
</tr>
<tr>
<td>6</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_I2C_INT</td>
<td>When an operation is complete on the I2C Master</td>
</tr>
<tr>
<td>7 (MSB)</td>
<td>MASK_I2CM_FIFO_INT_ENABLE_DONE</td>
<td></td>
</tr>
</tbody>
</table>

Returns
0 on success, -1 on an error

2.12.3.7 i2cm_is_interrupted

int8_t i2cm_is_interrupted ( uint8_t mask )
Check that an interrupt has been fired.
2.13 I²C Slave

The file `ft900_i2cs.h` contains the definitions for the I2C slave bus functions in the `libft900.a` library.

### 2.13.1 API Cross Reference

It utilises the following library APIs:

- `ft900_asm.h` – FT90X and FT93X assembler definitions
- `ft900_registers.h` – FT90X and FT93X register definitions

### 2.13.2 Function Documentation

#### 2.13.2.1 i2cs_init

```c
void i2cs_init ( uint8_t addr )
```

Call once, to initialise the slave and reset it to a known state.

**Parameters**

- `addr` Slave (read or write) address

#### 2.13.2.2 i2cs_read

```c
int8_t i2cs_read ( uint8_t * data, size_t size )
```

Read a specified number of bytes from the I2C Slave.

This transaction is orchestrated by the I2C Master. The number of bytes written may be less than the number requested if the master terminates the transaction early.

**Parameters**

- `data` Pointer to the byte buffer
- `size` Number of bytes to read

---

See also

- `i2cm_interrupt_enabled`

**Parameters**

- `mask` The bit pattern of interrupts to check

**Returns**

- 1 if the interrupt has been fired, 0 if the interrupt has not been fired, -1 otherwise

---

Copyright © Bridgetek Pte Ltd

Document Feedback

Product Page

Clearance No.: BRT#075

Document Reference No.: BRT_000118
[out] **data** Caller-allocated buffer to receive any bytes read.

[in] **size** The number of bytes to read.

**Returns**
0 on success, -1 on an error

### 2.13.2.3 i2cs_write

```c
int8_t i2cs_write ( const uint8_t * data,
                     size_t size
                 )
```

Write a specified number of bytes to an open device.

This transaction is orchestrated by the I2C Master. The number of bytes written may be less than the number requested if the master terminates the transaction early.

**Parameters**

- **[in] data** Buffer containing bytes to write.
- **[in] size** The number of bytes to write.

**Returns**
0 on success, -1 on an error
2.13.2.4 i2cs_get_status

uint8_t i2cs_get_status ( void )

Determine I2C Slave status.

Table 8- I2CS Status Mask

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask Name</th>
<th>Set when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (LSB)</td>
<td>MASK_I2CS_STATUS_RX_REQ</td>
<td>The slave controller head received data</td>
</tr>
<tr>
<td>1</td>
<td>MASK_I2CS_STATUS_TX_REQ</td>
<td>The slave controller is transmitter and requires data</td>
</tr>
<tr>
<td>2</td>
<td>MASK_I2CS_STATUS_SEND_FIN</td>
<td>The master has ended the receive operation</td>
</tr>
<tr>
<td>3</td>
<td>MASK_I2CS_STATUS_REC_FIN</td>
<td>The master has ended the transmit operation</td>
</tr>
<tr>
<td>4</td>
<td>MASK_I2CS_STATUS_BUS_ACTV</td>
<td>The bus is currently busy with an operation</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>MASK_I2CS_STATUS_DEV_ACTV</td>
<td>The slave controller is enabled</td>
</tr>
<tr>
<td>7 (MSB)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Returns

Returns the status byte of the I2C Slave Status register

2.13.2.5 i2cs_disable_interrupt

int8_t i2cs_disable_interrupt ( uint8_t mask )

Disable an interrupt.

Parameters

mask The bit pattern of interrupts to disable

Returns

0 on success, -1 on an error

2.13.2.6 i2cs_enable_interrupt

int8_t i2cs_enable_interrupt ( uint8_t mask )

Enable an interrupt.

Parameters

mask The bit pattern of interrupts to enable

Returns

0 on success, -1 on an error
2.14 I²S Audio

The file `ft900_i2s.h` contains the definitions for the I²S bus functions in the libft900.a library.

### 2.14.1 API Cross Reference

It utilises the following library APIs:

- `ft900_asm.h` – FT90X assembler definitions

Additional definitions are taken from:

- `ft900_registers.h` – FT90X register definitions

### 2.14.2 Enumeration Type Documentation

#### 2.14.2.1 i2s_mode_t

```c
enum i2s_mode_t
{
    i2s_mode_slave,    // I²S Slave
    i2s_mode_master    // I²S Master
};
```

#### 2.14.2.2 i2s_format_t

```c
enum i2s_format_t
{
    i2s_format_i2s,  // I²S format
    i2s_format_leftjust,  // Left justified format
    i2s_format_rightjust // Right justified format
};
```

#### 2.14.2.3 i2s_length_t

```c
enum i2s_length_t
{
    i2s_length_16,    // 16 bit data length
    i2s_length_20,    // 20 bit data length
    i2s_length_24,    // 24 bit data length
    i2s_length_32     // 32 bit data length
};
```
2.14.2.4 i2s_padding_t

enum i2s_padding_t

I2S padding definitions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2s_padding_0</td>
<td>No padding</td>
</tr>
<tr>
<td>i2s_padding_4</td>
<td>4 bits of padding</td>
</tr>
<tr>
<td>i2s_padding_8</td>
<td>8 bits of padding</td>
</tr>
<tr>
<td>i2s_padding_12</td>
<td>12 bits of padding</td>
</tr>
<tr>
<td>i2s_padding_16</td>
<td>16 bits of padding</td>
</tr>
</tbody>
</table>

2.14.2.5 i2s_bclk_div

enum i2s_bclk_div

I2S BCLK speed definitions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2s_bclk_div_1</td>
<td>Divide BCLK by 1</td>
</tr>
<tr>
<td>i2s_bclk_div_2</td>
<td>Divide BCLK by 2</td>
</tr>
<tr>
<td>i2s_bclk_div_3</td>
<td>Divide BCLK by 3</td>
</tr>
<tr>
<td>i2s_bclk_div_4</td>
<td>Divide BCLK by 4</td>
</tr>
<tr>
<td>i2s_bclk_div_6</td>
<td>Divide BCLK by 6</td>
</tr>
<tr>
<td>i2s_bclk_div_8</td>
<td>Divide BCLK by 8</td>
</tr>
<tr>
<td>i2s_bclk_div_12</td>
<td>Divide BCLK by 12</td>
</tr>
<tr>
<td>i2s_bclk_div_16</td>
<td>Divide BCLK by 16</td>
</tr>
<tr>
<td>i2s_bclk_div_24</td>
<td>Divide BCLK by 24</td>
</tr>
<tr>
<td>i2s_bclk_div_32</td>
<td>Divide BCLK by 32</td>
</tr>
<tr>
<td>i2s_bclk_div_48</td>
<td>Divide BCLK by 48</td>
</tr>
<tr>
<td>i2s_bclk_div_64</td>
<td>Divide BCLK by 64</td>
</tr>
</tbody>
</table>

2.14.2.6 i2s_mclk_div_t

enum i2s_mclk_div_t

I2S MCLK speed definitions.
## Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2s_mclk_div_1</td>
<td>Divide MCLK by 1</td>
</tr>
<tr>
<td>i2s_mclk_div_2</td>
<td>Divide MCLK by 2</td>
</tr>
<tr>
<td>i2s_mclk_div_3</td>
<td>Divide MCLK by 3</td>
</tr>
<tr>
<td>i2s_mclk_div_4</td>
<td>Divide MCLK by 4</td>
</tr>
<tr>
<td>i2s_mclk_div_6</td>
<td>Divide MCLK by 6</td>
</tr>
<tr>
<td>i2s_mclk_div_8</td>
<td>Divide MCLK by 8</td>
</tr>
<tr>
<td>i2s_mclk_div_12</td>
<td>Divide MCLK by 12</td>
</tr>
<tr>
<td>i2s_mclk_div_16</td>
<td>Divide MCLK by 16</td>
</tr>
<tr>
<td>i2s_mclk_div_24</td>
<td>Divide MCLK by 24</td>
</tr>
<tr>
<td>i2s_mclk_div_32</td>
<td>Divide MCLK by 32</td>
</tr>
<tr>
<td>i2s_mclk_div_48</td>
<td>Divide MCLK by 48</td>
</tr>
<tr>
<td>i2s_mclk_div_64</td>
<td>Divide MCLK by 64</td>
</tr>
</tbody>
</table>

### 2.14.2.7 i2s_bclk_per_channel_t

```c
enum i2s_bclk_per_channel_t
{
    i2s_bclk_per_channel_16,  // 16 BCLK per channel
    i2s_bclk_per_channel_32,  // 32 BCLK per channel
};
```

I2S BCLK cycles per channel, used in master mode only.

### 2.14.2.8 i2s_master_input_clk_t

```c
enum i2s_master_input_clk_t
{
    i2s_master_input_clk_22mhz,  // 22.5792 MHz Master clock
    i2s_master_input_clk_24mhz,  // 24.576 MHz Master clock
};
```

I2S master input clk frequency definitions.

### 2.14.2.9 i2s_rx_t

```c
enum i2s_rx_t
{
};
```

I2S RX definitions.
2.14.2.10  i2s_tx_t

eNum i2s_tx_t

I2S TX definitions.

---

2.14.3 Function Documentation

2.14.3.1 i2s_init

void i2s_init ( i2s_mode_t mode,
              i2s_length_t length,
              i2s_format_t format,
              i2s_padding_t padding,
              i2s_master_input_clk_t mclk_in,
              i2s_bclk_div bclk_div,
              i2s_mclk_div_t mclk_div,
              i2s_bclk_per_channel_t bclk_per_channel
          )

Call once, to initialise the peripheral and reset it to a known state.

**Parameters**

[in] **mode**  
The I2S Mode

[in] **length**  
The transfer length

[in] **format**  
The format of the transfer

[in] **padding**  
The number of padding bits that have been added to the transfer

[in] **mclk_in**  
The MCLK to use

[in] **bclk_div**  
The BCLK divider

[in] **mclk_div**  
The MCLK divider

[in] **bclk_per_channel**  
The number of BCLK per channel
2.14.3.2 i2s_start_rx

void i2s_start_rx ( void )
Start reception of the I2S Module.

2.14.3.3 i2s_start_tx

void i2s_start_tx ( void )
Start transmission of the I2S Module.

2.14.3.4 i2s_stop_rx

void i2s_stop_rx ( void )
Stop reception of the I2S Module.

2.14.3.5 i2s_stop_tx

void i2s_stop_tx ( void )
Stop transmission of the I2S Module.

2.14.3.6 i2s_read

size_t i2s_read ( uint8_t * data,
const size_t num_bytes
)
Reads x number of bytes from the I2S FIFO.

Parameters

[out] data Caller-allocated buffer to receive any bytes read.
[in] num_bytes Number of bytes to read from the FIFO.

2.14.3.7 i2s_write

size_t i2s_write ( const uint8_t * data,
const size_t num_bytes
)
Writes x number of bytes to the I2S FIFO.

Warning
Due to the hardware implementation of I2S, there is a performance hit when using this function with 24 bit input.

Parameters

[in] data Buffer containing bytes to write.
[in] num_bytes Number of bytes to write to the FIFO.
2.14.3.8 i2s_get_status

```c
uint16_t i2s_get_status ( void )
```

Determine I2S status.
Get the status of the interrupts on the I2S module.

**Returns**

Returns the status of the I2S peripheral.
The copy of the Interrupt Status Register

2.14.3.9 i2s_get_rx_count

```c
uint16_t i2s_get_rx_count ( void )
```

Get the receive count of the I2S Module.

**Returns**

The number of receptions that have been made

2.14.3.10 i2s_get_tx_count

```c
uint16_t i2s_get_tx_count ( void )
```

Get the transmit count of the I2S Module.

**Returns**

The number of transmissions that have been made

2.14.3.11 i2s_disable_int

```c
void i2s_disable_int ( uint16_t mask )
```

Disable interrupts on the I2S module.

**Parameters**

- **mask** The mask of bits to disable

**See also**

i2s_enable_int

2.14.3.12 i2s_enable_int

```c
void i2s_enable_int ( uint16_t mask )
```

Enable interrupts on the I2S module.

---

**Table 9- I2S Interrupt Enable Mask**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask Name</th>
<th>Interrupts when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (LSB)</td>
<td>MASK_I2S_IE_FIFO_TX_UNDER</td>
<td>Transmit FIFO has underflowed</td>
</tr>
<tr>
<td>1</td>
<td>MASK_I2S_IE_FIFO_TX_EMPTY</td>
<td>Transmit FIFO is empty</td>
</tr>
<tr>
<td>2</td>
<td>MASK_I2S_IE_FIFO_TX_HALF_FULL</td>
<td>Transmit FIFO is half empty</td>
</tr>
<tr>
<td>3</td>
<td>MASK_I2S_IE_FIFO_TX_FULL</td>
<td>Transmit FIFO is full</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MASK_I2S_IE_FIFO_TX_OVER</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MASK_I2S_IE_FIFO_RX_UNDER</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MASK_I2S_IE_FIFO_RX_EMPTY</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MASK_I2S_IE_FIFO_RX_HALF_FULL</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MASK_I2S_IE_FIFO_RX_FULL</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MASK_I2S_IE_FIFO_RX_OVER</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>(MSB) -</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

**mask** The mask of bits to enable

### 2.14.3.13 i2s_clear_int_flag

```c
void i2s_clear_int_flag ( uint16_t mask )
```

Clear interrupt flags on the I2S module.

**Parameters**

**mask** The mask of bits to clear

### 2.14.3.14 i2s_is_interrupted

```c
int8_t i2s_is_interrupted ( uint16_t mask )
```

Check if an interrupt has been fired.

**Warning**

This function will clear the current interrupts you are checking for.

**Parameters**

**mask** The mask of interrupts to check for

**Returns**

1 when an interrupt has fired, 0 otherwise.

**See also**

i2s_enable_int
2.15 SPI Bus

The file ft900_spi.h contains the definitions for the SPI Master and Slave bus functions in the libft900.a library.

2.15.1 API Cross Reference

Additional definitions are taken from:

ft900_registers.h – FT90X and FT93X register definitions

2.15.2 Enumeration Type Documentation

2.15.2.1 enum spi_clock_mode_t

The SPI mode.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_mode_0</td>
<td>CPOL = 0, CPHA = 0</td>
</tr>
<tr>
<td>spi_mode_1</td>
<td>CPOL = 0, CPHA = 1</td>
</tr>
<tr>
<td>spi_mode_2</td>
<td>CPOL = 1, CPHA = 0</td>
</tr>
<tr>
<td>spi_mode_3</td>
<td>CPOL = 1, CPHA = 1</td>
</tr>
</tbody>
</table>

2.15.2.2 enum spi_option_t

SPI Options.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_option_fifo</td>
<td>Enable or disable the FIFO</td>
</tr>
<tr>
<td>spi_option_fifo_size</td>
<td>Set the size of the FIFO</td>
</tr>
<tr>
<td>spi_option_fifo_receive_trigger</td>
<td>Set the FIFO trigger level</td>
</tr>
<tr>
<td>spi_option_force_ss_assertions</td>
<td>Force SS to go low in assert</td>
</tr>
<tr>
<td>spi_option_bus_width</td>
<td>Set the SPI bus width</td>
</tr>
<tr>
<td>spi_option_multi_receive</td>
<td>Set the SPI device to clock in data without loading it into the RX FIFO</td>
</tr>
<tr>
<td>spi_option_fifo_transmit_trigger</td>
<td>Set the FIFO transmit trigger level</td>
</tr>
<tr>
<td>spi_option_baud_factor</td>
<td>Applicable in FT93X and FT90X Revision C. SCK frequency is determined by BAUD FACTOR ( divided by 2 to divided by 256)</td>
</tr>
<tr>
<td>spi_option_fast_spi</td>
<td>Applicable ONLY in FT90X Revision C. The master clock is set to CLK (system clock)/2</td>
</tr>
<tr>
<td>spi_option_change_spi_rate</td>
<td>SCK frequency is determined by CHG_SPI and SPR[0:2] bits (CLK gets divided 2 to divided by 511)</td>
</tr>
<tr>
<td>spi_option_ignoreIncoming</td>
<td>Not supported as of now.</td>
</tr>
</tbody>
</table>

2.15.2.3 enum spi_dir_t

SPI Direction.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_dir_slave</td>
<td>The SPI Device is in Slave mode</td>
</tr>
<tr>
<td>spi_dir_master</td>
<td>The SPI Device is in Master mode</td>
</tr>
</tbody>
</table>

2.15.2.4 enum spi_fifo_size_t

SPI FIFO size.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_fifo_size_16</td>
<td>Use a 16 level FIFO</td>
</tr>
<tr>
<td>spi_fifo_size_64</td>
<td>Use a 64 Byte FIFO</td>
</tr>
</tbody>
</table>

2.15.2.5 enum spi_width_t

SPI Data Bus Width.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_width_1bit</td>
<td>The SPI Device is working in 1 bit wide mode (i.e. 4 wire SPI)</td>
</tr>
<tr>
<td>spi_width_2bit</td>
<td>The SPI Device is working in 2 bit wide mode</td>
</tr>
<tr>
<td>spi_width_4bit</td>
<td>The SPI Device is working in 4 bit wide mode</td>
</tr>
</tbody>
</table>

2.15.2.6 enum spi_ss_assertions_t

SS Assertion control.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_ss_assertions_force</td>
<td>SS will reflect the status of SPISS</td>
</tr>
<tr>
<td>spi_ss_assertions_auto</td>
<td>SS will go low during transmissions if selected</td>
</tr>
</tbody>
</table>

2.15.2.7 enum spi_interrupt_t

SPI Interrupts

NOTE: Call spi_is_interrupted() to clear interrupt flags.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_interrupt_transmit_empty</td>
<td>Either the FIFO or the data register are empty</td>
</tr>
</tbody>
</table>
spi_interrupt_data_ready | A transmission or reception was completed or the FIFO was filled to a trigger level.
---|---
spi_interrupt_transmit_1bit_complete | Transmission was complete when using the SPI1BIT method.
spi_interrupt_fault | The SPI device was asserted when in Master mode.

### 2.15.3 Function Documentation

#### 2.15.3.1 spi_init

```c
int8_t spi_init ( ft900_spi_regs_t * dev,
    spi_dir_t dir,
    spi_clock_mode_t clock_mode,
    uint16_t div
)
```

Initialise the SPI device.

**Parameters**

- **dev** the device to use
- **dir** The direction for the device to work in (Master/Slave)
- **clock_mode** The SPI Clock mode to use
- **div** The clock divider to use (4,8,16,32,64,128,256,512)

**Returns**

0 on a success or -1 for a failure

**Warning**

This will reset the peripheral and all options

#### 2.15.3.2 spi_open

```c
int8_t spi_open ( ft900_spi_regs_t* dev,
    uint8_t num
)
```

Select a device to start communicating with.

**Parameters**

- **dev** the device to use
- **num** The device to select

**Returns**

0 on a success or -1 for a failure
2.15.3.3 spi_close

```c
int8_t spi_close ( ft900_spi_regs_t * dev,
                    uint8_t num
                );
```

Stop communicating with a certain device.

**Parameters**
- `dev` the device to use
- `num` The device to select

**Returns**
- 0 on a success or -1 for a failure

2.15.3.4 spi_read

```c
int32_t spi_read ( ft900_spi_regs_t * dev,
                    uint8_t b
                );
```

Reads a byte from the SPI device.

**Parameters**
- `dev` the device to use
- `b` A variable to store the byte

**Returns**
- The number of bytes read or -1 for a failure

2.15.3.5 spi_readn

```c
int32_t spi_readn ( ft900_spi_regs_t * dev,
                     uint8_t * b,
                     size_t len
                 );
```

Reads several bytes from the SPI device.

**Parameters**
- `dev` the device to use
- `b` A pointer to the array to read into
- `len` The number of bytes to read

**Returns**
- The number of bytes read or -1 for a failure
2.15.3.6 spi_write

```c
int32_t spi_write ( ft900_spi_regs_t * dev,
                    uint8_t b )
```

Writes a byte to the SPI device.

**Parameters**
- `dev`: the device to use
- `b`: The byte to send

**Returns**
The number of bytes written or -1 for a failure

2.15.3.7 spi_writen

```c
int32_t spi_writen ( ft900_spi_regs_t * dev,
                     const uint8_t * b,
                     size_t len )
```

Writes several bytes to the SPI device.

**Parameters**
- `dev`: the device to use
- `b`: A pointer to the array to send
- `len`: The number of bytes to write

**Returns**
The number of bytes written or -1 for a failure

2.15.3.8 spi_xchange

```c
int32_t spi_xchange ( ft900_spi_regs_t * dev,
                      uint8_t b,
                      uint8_t c )
```

Exchange a byte to the SPI device. Supports single channel mode only.

**Parameters**
- `dev`: the device to use
- `b`: A variable to send the byte
- `c`: A variable to store the byte

**Returns**
The number of bytes exchanged or -1 for failure.
2.15.3.9 spi_xchangen

```c
int32_t spi_xchangen ( ft900_spi_regs_t * dev,
        uint8_t * binp,
        uint8_t * bout,
        size_t len )
```

Exchange several bytes to the SPI device. Supports single channel mode only.

**Parameters**
- `dev` the device to use
- `binp` A pointer to the array to send
- `bout` A pointer to the array to receive
- `len` The number of bytes to exchange

**Returns**

2.15.3.10 The number of bytes exchanged or -1 for a failure

```c
uint8_t spi_status ( ft900_spi_regs_t * dev )
```

Return the status register.

**Table 10- SPI Status flags**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>When set to 1</th>
<th>When set to 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AUTOSS</td>
<td>Auto SS Assertions Enabled</td>
<td>SSO always shows contents of SSCR</td>
</tr>
<tr>
<td>1</td>
<td>RXFULL</td>
<td>Receiver FIFO is full</td>
<td>Receiver FIFO not is full</td>
</tr>
<tr>
<td>2</td>
<td>EMPTY</td>
<td>TX FIFO or TX register empty</td>
<td>TX FIFO or TX register not empty</td>
</tr>
<tr>
<td>3</td>
<td>IDLE</td>
<td>SPI Device Idle</td>
<td>Transmission in progress</td>
</tr>
<tr>
<td>4</td>
<td>FAULT</td>
<td>Mode Fault (SS Low in Master mode)</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1BITTX</td>
<td>End of TX from SPDR BIS Register</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>WCOL</td>
<td>Data Register Write Collision occurred</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>IRQ</td>
<td>An interrupt occurred</td>
<td>-</td>
</tr>
</tbody>
</table>

**Parameters**
- `dev` The device to use

**Returns**
A copy of the SPISTAT register

2.15.3.11 spi_option

```c
int8_t spi_option ( ft900_spi_regs_t * dev,
```
Control the SPI device.
This function will set various options for the driver.

**Table 11- SPIM Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>spim_option_fifo</td>
<td>Enable or disable the FIFO</td>
<td>0 = Disabled (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Enabled [Note 1]</td>
</tr>
<tr>
<td>spim_option_fifo_size</td>
<td>Set the size of the FIFO</td>
<td>16 = 16 Byte FIFO (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 = Byte FIFO</td>
</tr>
<tr>
<td>spim_option_fifo_receive_trigger</td>
<td>Set the FIFO trigger level</td>
<td>For 16 Byte FIFOs: 1, 4, 8, 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For 64 Byte FIFOs: 1, 16, 32, 56</td>
</tr>
<tr>
<td>spim_option_force_ss_assertions</td>
<td>Force SS to go low in assert</td>
<td>0 = Automatic Assertions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Force Assertions (Default)</td>
</tr>
<tr>
<td>spim_option_bus_width</td>
<td>Set the SPI bus width</td>
<td>1 = Single (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Dual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = Quad</td>
</tr>
<tr>
<td>spim_option_dual_quad_direction</td>
<td>Set the multi-bit direction</td>
<td>0 = Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Write</td>
</tr>
</tbody>
</table>

Note 1: Enabling the FIFO will cause the driver to clear its contents.

**Parameters**

- `dev` The device to use
- `opt` The option to configure
- `val` The value to use

**Returns**

0 on a success or -1 for a failure

### 2.15.3.12 `spi_disable_interrupt`

```c
int8_t spi_disable_interrupt ( ft900_spi_regs_t * dev,
                                spi_interrupt_t interrupt
                              )
```

Disables an interrupt for the SPI device.

**Parameters**

- `dev` The device to use
- `interrupt` The interrupt to disable

**Returns**
0 on a success or -1 for a failure

See also
spim_disable_interrupts_globally

2.15.3.13  **spi_disable_interrupts_globally**

    int8_t spi_disable_interrupts_globally ( ft900_spi_regs_t* dev )

Disables the SPI device from generating an interrupt.

**Parameters**

- `dev` the device to use

**Returns**

0 on a success or -1 for a failure

2.15.3.14  **spi_enable_interrupt**

    int8_t spi_enable_interrupt ( ft900_spi_regs_t* dev, 
                                  spi_interrupt_t interrupt  
                            )

Enables an interrupt for the SPI device.

**Parameters**

- `dev` the device to use
- `interrupt` The interrupt to enable

**Returns**

0 on a success or -1 for a failure

See also
spim_enable_interrupts_globally

2.15.3.15  **spi_enable_interrupts_globally**

    int8_t spi_enable_interrupts_globally ( ft900_spi_regs_t* dev )

Enables the SPI device to generate an interrupt.

**Parameters**

- `dev` the device to use

**Returns**

0 on a success or -1 for a failure

2.15.3.16  **spi_is_interrupted**

    int8_t spi_is_interrupted ( ft900_spi_regs_t* dev, 
                                spi_interrupt_t interrupt  
                          )

Disables an interrupt for the QSPI device.

**Parameters**
dev The device to use

interrupt The interrupt to check

Returns
1 when interrupted, 0 when not interrupted, -1 otherwise

See also
spi_disable_interrupts_globally

2.15.3.17 spi_uninit

int8_t spi_uninit ( ft900_spi_regs_t* dev )

Disable the SPI device.

Parameters
dev the device to use

Returns
0 on a success or -1 for a failure

2.16 CANBus

The file ft900_can.h contains the definitions for the CANBus functions in the libft900.a library.

2.16.1 API Cross Reference

Additional definitions are taken from:
ft900_registers.h – FT90X register definitions

2.16.2 Enumeration Type Documentation

2.16.2.1 can_mode_t

denum can_mode_t

The mode of the CAN device.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_mode_normal</td>
<td>The CAN Device is operating normally</td>
</tr>
<tr>
<td>can_mode_listen</td>
<td>The CAN Device is in listen-only mode and will not allow data transmission or send ACKs on the bus</td>
</tr>
</tbody>
</table>

2.16.2.2 can_type_t

denum can_type_t

CAN message type.
### Enumerator

<table>
<thead>
<tr>
<th>can_type_standard</th>
<th>CAN2.0A Standard Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_type_extended</td>
<td>CAN2.0B Extended Frame</td>
</tr>
</tbody>
</table>

**2.16.2.3 can_filter_mode_t**

```c
enum can_filter_mode_t
{
  can_filter_mode_single,  // Single Filter mode
  can_filter_mode_dual     // Dual Filter mode
};
```

The mode of the filter.

### Enumerator

<table>
<thead>
<tr>
<th>can_filter_mode_single</th>
<th>Single Filter mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_filter_mode_dual</td>
<td>Dual Filter mode</td>
</tr>
</tbody>
</table>

**2.16.2.4 can_arbitration_lost_t**

```c
enum can_arbitration_lost_t
{
};
```

The location of where arbitration was lost.

### Enumerator

<table>
<thead>
<tr>
<th>can_arbitration_lost_id28_11</th>
<th>Arbitration was lost on the 12th bit of the ID (29th bit extended)</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_arbitration_lost_id27_10</td>
<td>Arbitration was lost on the 11th bit of the ID (28th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id26_9</td>
<td>Arbitration was lost on the 10th bit of the ID (27th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id25_8</td>
<td>Arbitration was lost on the 9th bit of the ID (26th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id24_7</td>
<td>Arbitration was lost on the 8th bit of the ID (25th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id23_6</td>
<td>Arbitration was lost on the 7th bit of the ID (24th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id22_5</td>
<td>Arbitration was lost on the 6th bit of the ID (23th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id21_4</td>
<td>Arbitration was lost on the 5th bit of the ID (22th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id20_3</td>
<td>Arbitration was lost on the 4th bit of the ID (21th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id19_2</td>
<td>Arbitration was lost on the 3rd bit of the ID (20th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id18_1</td>
<td>Arbitration was lost on the 2nd bit of the ID (19th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_id17_0</td>
<td>Arbitration was lost on the 1st bit of the ID (18th bit extended)</td>
</tr>
<tr>
<td>can_arbitration_lost_srtr_rtr</td>
<td>Arbitration was lost on the SRTR/RTR bit</td>
</tr>
<tr>
<td>can_arbitration_lost_id16</td>
<td>Arbitration was lost on the 17th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id15</td>
<td>Arbitration was lost on the 16th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id14</td>
<td>Arbitration was lost on the 15th bit of the extended ID</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>can_arbitration_lost_id13</td>
<td>Arbitration was lost on the 14th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id12</td>
<td>Arbitration was lost on the 13th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id11</td>
<td>Arbitration was lost on the 12th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id10</td>
<td>Arbitration was lost on the 11th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id9</td>
<td>Arbitration was lost on the 10th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id8</td>
<td>Arbitration was lost on the 9th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id7</td>
<td>Arbitration was lost on the 8th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id6</td>
<td>Arbitration was lost on the 7th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id5</td>
<td>Arbitration was lost on the 6th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id4</td>
<td>Arbitration was lost on the 5th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id3</td>
<td>Arbitration was lost on the 4th bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id2</td>
<td>Arbitration was lost on the 3rd bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id1</td>
<td>Arbitration was lost on the 2nd bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_id0</td>
<td>Arbitration was lost on the 1st bit of the extended ID</td>
</tr>
<tr>
<td>can_arbitration_lost_rtr</td>
<td>Arbitration was lost on the RTR bit</td>
</tr>
<tr>
<td>can_arbitration_lost_invalid</td>
<td>No valid arbitration could be found</td>
</tr>
</tbody>
</table>

### 2.16.2.5 can_interrupt_t

```c
enum can_interrupt_t

CAN Peripheral Interrupts.

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_interrupt_data_overrun</td>
</tr>
<tr>
<td>can_interrupt_bus_error</td>
</tr>
<tr>
<td>can_interrupt_transmit</td>
</tr>
<tr>
<td>can_interrupt_receive</td>
</tr>
<tr>
<td>can_interrupt_error_passive</td>
</tr>
<tr>
<td>can_interrupt_error_warning</td>
</tr>
<tr>
<td>can_interrupt_arbitration_lost</td>
</tr>
<tr>
<td>can_interrupt_data_overload</td>
</tr>
</tbody>
</table>
```

This enum for FT90X
2.16.2.6 can_rtr_t

enum can_rtr_t

Remote transfer request flag.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>can_rtr_none</td>
<td>No RTR</td>
</tr>
<tr>
<td>can_rtr_remote_request</td>
<td>RTR Set</td>
</tr>
</tbody>
</table>

2.16.3 Function Documentation

2.16.3.1 can_init

int8_t can_init ( ft900_can_regs_t * dev, 
                 can_mode_t mode, 
                 const can_time_config_t * timeconfig 
)

Initialise the CAN device.

Parameters

- **dev**  A pointer to the device to use
- **mode** The mode that this CAN device should be in (normal or listen)
- **timeconfig** The configuration struct that defines the timing of the device

Returns

0 on a success, -1 otherwise

2.16.3.2 can_open

int8_t can_open ( ft900_can_regs_t * dev )

Open the CAN device for reading and writing.

Parameters

- **dev** A pointer to the device to use

Returns

0 on a success, -1 otherwise

2.16.3.3 can_close

int8_t can_close ( ft900_can_regs_t * dev )

Close the CAN device for reading and writing.

Parameters
**dev** A pointer to the device to use

**Returns**

0 on a success, -1 otherwise

### 2.16.3.4 can_filter

```c
int8_t can_filter ( ft900_can_regs_t* dev,
                    can_filter_mode_t filtmode,
                    uint8_t filternum,
                    const can_filter_t* filter
)
```

Set up a filter for the CAN device.
Set up the CAN device to filter for specific criteria.
The filter can work in two modes: single or dual. Depending on which mode the filter is in certain types of information can be used, as shown in the table below.

**Table 12- CAN mode and message filters**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Message Type</th>
<th>ID</th>
<th>RTR</th>
<th>Data[0]</th>
<th>Data[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Standard</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Single</td>
<td>Extended</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dual</td>
<td>Standard</td>
<td>Yes</td>
<td>Yes</td>
<td>Filter 0 Only</td>
<td>No</td>
</tr>
<tr>
<td>Dual</td>
<td>Extended</td>
<td>Only bits 13 to 28</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Any field which is not used in a certain configuration will be ignored.

**Parameters**

- **dev** A pointer to the device to use
- **filtmode** The mode that the filters should be in (single or dual)
- **filternum** The number of filter to use. When in single mode, only 0 will work.
- **filter** A pointer to the configuration to use

**Returns**

0 on a success, -1 otherwise

**Warning**

This command only works when the CAN device is closed

**See also**

- can_filter_t

### 2.16.3.5 can_read

```c
int8_t can_read ( ft900_can_regs_t* dev,
                  can_msg_t* msg
)
Receive a message from the CAN Bus.
This function will take the first available message from the Receive FIFO.

**Parameters**

- **dev** A pointer to the device to use
- **msg** The struct to pack the message into

**Returns**

0 on a success, -1 otherwise

**Warning**

This command only works when the CAN device is open

This function will automatically clear the Receive interrupt flag and increment the Receive message counter.

### 2.16.3.6 can_write

```c
int8_t can_write ( ft900_can_regs_t * dev,
                   const can_msg_t * msg
                 )
```

Send a message on the CAN Bus.
This function will accept a can_msg_t and pack it into a format to be fed into the CAN Transmit FIFO.

**Parameters**

- **dev** A pointer to the device to use
- **msg** The message to send

**Returns**

0 on a success, -1 otherwise

**Warning**

This command only works when the CAN device is open

### 2.16.3.7 can_abort

```c
int8_t can_abort ( ft900_can_regs_t * dev )
```

Abort the transmission of messages on the CAN Bus.
This function will cause the CAN device to abort transmission on the CAN Bus. After the transmission of the current message on the Bus, no further transmissions will occur (including retransmissions for erroneous messages).

**Parameters**

- **dev** A pointer to the device to use

**Returns**

0 on a success, -1 otherwise
2.16.3.8 can_status

```c
uint8_t can_status ( ft900_can_regs_t * dev )
```

Query the status register.
The return value is a bit-mask with the following format:

**Table 13- CAN Status Bit Mask**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>Set when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (MSB)</td>
<td>RX_BUF_STS</td>
<td>Receive Buffer Status</td>
<td>At least one message in the receive FIFO</td>
</tr>
<tr>
<td>6</td>
<td>OVRN_STS</td>
<td>Data Overrun Status</td>
<td>The receive FIFO has encountered an overrun</td>
</tr>
<tr>
<td>5</td>
<td>TX_BUF_STS</td>
<td>Transmit Buffer Status</td>
<td>The CPU is able to write to the transmit FIFO</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RX_STS</td>
<td>Receive Status</td>
<td>CAN is currently receiving a message</td>
</tr>
<tr>
<td>2</td>
<td>TX_STS</td>
<td>Transmit Status</td>
<td>CAN is currently transmitting a message</td>
</tr>
<tr>
<td>1</td>
<td>ERR_STS</td>
<td>Error Status</td>
<td>One CAN error counter has reached warning (96)</td>
</tr>
<tr>
<td>0 (LSB)</td>
<td>BUS_OFF_STS</td>
<td>Bus Off Status</td>
<td>The CAN device is in a bus off state</td>
</tr>
</tbody>
</table>

**Parameters**

- `dev` A pointer to the device to use

**Returns**

A bit-mask of the current status or 0 for an unknown device.

2.16.3.9 can_available

```c
uint8_t can_available ( ft900_can_regs_t * dev )
```

Return how many messages are available in the receive FIFO.

**Parameters**

- `dev` A pointer to the device to use

**Returns**

The number of messages available in the receive FIFO.

2.16.3.10 can_rx_error_count

```c
uint8_t can_rx_error_count ( ft900_can_regs_t * dev )
```

Get the current number of receive errors reported by the CAN device.

**Parameters**

- `dev` A pointer to the device to use

**Returns**

...
2.16.3.11 \texttt{can\_tx\_error\_count}

\begin{verbatim}
uint8_t can_tx_error_count ( ft900_can_regs_t * dev )
\end{verbatim}

Get the current number of transmit errors reported by the CAN device.

When the transmit error counter exceeds limit of 255, the Bus Status bit in the Status register is set to logic 1 (bus off), the CAN controller set reset mode, and if enabled an error warning interrupt is generated. The transmit error counter is then set to 127 and receive error counter is cleared.

\textbf{Parameters}

- \emph{dev} A pointer to the device to use

\textbf{Returns}

The current number of receive errors (0 - 255) or 0 for an unknown device.

2.16.3.12 \texttt{can\_ecode}

\begin{verbatim}
uint8_t can_ecode ( ft900_can_regs_t * dev )
\end{verbatim}

Get the current value of the ECC (Error Code Capture) register.

This function will return the value of the ECC (Error Code Capture) register. This register holds the error code for the \textit{LAST} bus error that occurred on the CAN network.

The return value is a bit-mask with the following format:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|p{7cm}|}
\hline
\textbf{Bit} & \textbf{Name} & \textbf{Description} & \textbf{Set when...} \\
\hline
7 (MSB) & RX_WRN & Receive Warning & The number of receive errors is $\geq 96$ \\
6 & TX_WRN & Transmit Warning & The number of transmit errors is $\geq 96$ \\
5 & ERR_DIR & Direction & The error occurred on reception. \\
4 & ACK_ERR & Acknowledgement Error & An ACK Error occurred \\
3 & FRM_ERR & Form Error & A Form Error occurred \\
2 & CRC_ERR & CRC Error & A CRC Error occurred \\
1 & STF_ERR & Stuff Error & A Bit Stuffing Error occurred \\
0 (LSB) & BIT_ERR & Bit Error & A Bit Error occurred \\
\hline
\end{tabular}
\end{table}

\textbf{Parameters}

- \emph{dev} A pointer to the device to use

\textbf{Returns}

The value of the ECC (Error Code Capture) register or 0 for an unknown device.

2.16.3.13 \texttt{can\_arbitration\_lost\_location}

\begin{verbatim}
can_arbitration_lost_t can_arbitration_lost_location ( ft900_can_regs_t * dev )
\end{verbatim}
Get the location where arbitration was lost.

**Parameters**

`dev` A pointer to the device to use

**Returns**

The location where arbitration was lost

### 2.16.3.14 can_enable_interrupt

```c
int8_t can_enable_interrupt ( ft900_can_regs_t * dev,
                                can_interrupt_t interrupt
                        )
```

Enable an Interrupt.

Enable the CAN device to generate an interrupt. The value of mask is a bit-mask with the following format:

**Table 15- CAN Interrupt Enable Mask**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (MSB)</td>
<td></td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ARB_LOST</td>
<td>Arbitration Lost Interrupt</td>
<td>Arbitration is lost</td>
</tr>
<tr>
<td>5</td>
<td>ERR_WARN</td>
<td>Error Warning Interrupt</td>
<td>Changes in ES or BS of the Status register</td>
</tr>
<tr>
<td>4</td>
<td>ERR_PSV</td>
<td>Error Passive Interrupt</td>
<td>Bus enters or exits a passive state</td>
</tr>
<tr>
<td>3</td>
<td>RX</td>
<td>Receive Interrupt</td>
<td>A message is received on CAN</td>
</tr>
<tr>
<td>2</td>
<td>TX</td>
<td>Transmit Interrupt</td>
<td>A message is successfully received on CAN</td>
</tr>
<tr>
<td>1</td>
<td>BUS_ERR</td>
<td>Bus Error Interrupt</td>
<td>A bus error occurred when transmitting\receiving</td>
</tr>
<tr>
<td>0 (LSB)</td>
<td>DATA_OVRN</td>
<td>Data Overrun Interrupt</td>
<td>A receive FIFO overrun occurred</td>
</tr>
</tbody>
</table>

**Parameters**

`dev` A pointer to the device to use

`interrupt` The interrupt to enable

**Returns**

0 on a success, -1 otherwise

**Warning**

This command only works when the CAN device is closed

### 2.16.3.15 can_disable_interrupt

```c
int8_t can_disable_interrupt ( ft900_can_regs_t * dev,
```


can_interrupt_t interrupt
}

Disable an Interrupt.
Disable the CAN device from generating an interrupt.

**Parameters**
- `dev` A pointer to the device to use
- `interrupt` The interrupt to disable

**Returns**
- 0 on a success, -1 otherwise

**Warning**
This command only works when the CAN device is closed

### 2.16.3.16 can_is_interrupted

```c
int8_t can_is_interrupted ( ft900_can_regs_t* dev,
                          can_interrupt_t interrupt )
```

Query the Interrupt register.
Query the Interrupt register in order to determine what caused the interrupt.

**Parameters**
- `dev` A pointer to the device to use
- `interrupt` The interrupt to check

**Warning**
This function clears the interrupt bit so that it does not fire constantly

**Returns**
- 0 when the interrupt hasn't been fired, 1 when the interrupt has fired and -1 otherwise

### 2.16.4 Variable Documentation

- `const can_time_config_t g_can125kbaud`
  Configuration for 125 kBaud at `fcpu` = 100 MHz
- `const can_time_config_t g_can1Mbaud`
  Configuration for 1 MBaud at `fcpu` = 100 MHz
- `const can_time_config_t g_can250kbaud`
  Configuration for 250 kBaud at `fcpu` = 100 MHz
- `const can_time_config_t g_can500kbaud`
  Configuration for 500 kBaud at `fcpu` = 100 MHz
2.17 Camera interface

The file `ft900_cam.h` contains the definitions for the camera bus functions in the `libft900.a` library.

2.17.1 API Cross Reference

It utilises the following library APIs:

- `ft900_asm.h` – FT9X assembler definitions
- `ft900_registers.h` – FT9X register definitions

2.17.2 Enumeration Type Documentation

2.17.2.1 cam_clock_pol_t

```c
enum cam_clock_pol_t
{
    cam_clock_pol_falling = Sample data on a falling PCLK edge,
    cam_clock_pol_raising = Sample data on a raising PCLK edge,
}
```

2.17.2.2 cam_trigger_mode_t

```c
enum cam_trigger_mode_t
{
    cam_trigger_mode_0 = VD = L, HD = L,
    cam_trigger_mode_1 = VD = L, HD = H,
    cam_trigger_mode_2 = VD = H, HD = L,
    cam_trigger_mode_3 = VD = H, HD = H,
}
```

2.17.3 Function Documentation

2.17.3.1 cam_init

```c
int8_t cam_init ( cam_trigger_mode_t triggers,
    cam_clock_pol_t clkpol
)
```

Initialise the Camera interface.

**Parameters**

- `triggers` The VD/HD levels to trigger on
clkpol    The clock polarity of the input

Returns
0 on success, -1 otherwise

2.17.3.2 cam_available

uint16_t cam_available ( void )

Check how many bytes are available on the FIFO.

Returns
The number of bytes available

2.17.3.3 cam_start

int8_t cam_start ( uint16_t bytes )

Start capturing data.

Parameters
bytes The number of bytes to capture

Returns
0 on success, -1 otherwise

2.17.3.4 cam_stop

int8_t cam_stop ( void )

Stop capturing data.

Returns
0 on success, -1 otherwise

2.17.3.5 cam_set_threshold

int8_t cam_set_threshold ( uint16_t n )

Set the threshold for when the camera interrupt fires.

Parameters
n The number of bytes to fill the FIFO with before the interrupt fires (this must be a multiple of 4)

Returns
0 on success, -1 otherwise

2.17.3.6 cam_readn

uint16_t cam_readn ( uint8_t * b, size_t len )

Read a number of bytes from the FIFO.

Parameters
b A pointer to read the data into

cam_read

len The number of bytes to read from the FIFO (this must be a multiple of 4)

Returns
The number of bytes read, 0 otherwise

2.17.3.7 cam_flush

void cam_flush ( void )

Empty out the camera buffer.

2.17.3.8 cam_total_read

uint16_t cam_total_read ( void )

Check how many bytes have been read by the Camera Interface.

Returns
2.17.3.9 The number of bytes read cam_enable_interrupt

int8_t cam_enable_interrupt ( void )

Enable the threshold interrupt.

Returns
0 on success, -1 otherwise

2.17.3.10 cam_disable_interrupt

int8_t cam_disable_interrupt ( void )

Disable the threshold interrupt.

Returns
0 on success, -1 otherwise

2.17.3.11 cam_is_interrupted

int8_t cam_is_interrupted ( void )

Check that an interrupt has occurred.

Returns
0 when the interrupt hasn't been fired, 1 when the interrupt has fired and -1 otherwise
2.18 Pulse Width Modulation

The file `ft900_pwm.h` contains the definitions for the Pulse Width Modulation functions in the `libft900.a` library.

2.18.1 API Cross Reference

Additional definitions are taken from:

`ft900_registers.h` – FT90X and FT93X register definitions

2.18.2 Enumeration Type Documentation

2.18.2.1 `pwm_restore_t`

```
enum pwm_restore_t

PWM restore state.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_restore_disable</td>
<td>Do not restore the setup state on wrap around</td>
</tr>
<tr>
<td>pwm_restore_enable</td>
<td>Do not restore the setup state on wrap around</td>
</tr>
</tbody>
</table>

2.18.2.2 `pwm_state_t`

```
enum pwm_state_t

```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_state_low</td>
<td>Setup as low</td>
</tr>
<tr>
<td>pwm_state_high</td>
<td>Setup as high</td>
</tr>
</tbody>
</table>

2.18.2.3 `pwm_trigger_t`

```
enum pwm_trigger_t

```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_trigger_disabled</td>
<td>Do not trigger</td>
</tr>
<tr>
<td>pwm_trigger_positive_edge</td>
<td>Trigger on a positive edge</td>
</tr>
<tr>
<td>pwm_trigger_negative_edge</td>
<td>Trigger on a negative edge</td>
</tr>
<tr>
<td>pwm_trigger_any_edge</td>
<td>Trigger on any edge</td>
</tr>
</tbody>
</table>

2.18.3 Function Documentation

2.18.3.1 `pwm_init`

```c
int8_t pwm_init ( uint8_t prescaler,
```
Initialise the PWM subsystem.

**Parameters**

- **prescaler** The prescaler for the PWM subsystem
- **maxcount** The maximum count of the 16 bit master counter
- **shots** The number of loops the PWM subsystem will make, 0 is infinity

**Returns**

On success a 0, otherwise -1

### 2.18.3.2 pwm_enable

```c
int8_t pwm_enable ( void )
```

Enable the PWM subsystem.

**Returns**

On success a 0, otherwise -1

### 2.18.3.3 pwm_disable

```c
int8_t pwm_disable ( void )
```

Disable the PWM subsystem.

**Returns**

On success a 0, otherwise -1

### 2.18.3.4 pwm_add_toggle

```c
int8_t pwm_add_toggle ( uint8_t channel,
                       uint8_t toggle )
```

Add a toggle to a specific PWM channel.

**Parameters**

- **channel** The channel to add the toggle to
- **toggle** The channel to toggle on

**Returns**

On success a 0, otherwise -1

### 2.18.3.5 pwm_remove_toggle

```c
int8_t pwm_remove_toggle ( uint8_t channel,
                           uint8_t toggle )
```
Remove a toggle to a specific PWM channel.

**Parameters**

- **channel** The channel to remove the toggle from
- **toggle**  The channel to remove the toggle of

**Returns**
On success a 0, otherwise -1

### 2.18.3.6 pwm_compare

```c
int8_t pwm_compare ( uint8_t channel,
                      uint16_t value
                  )
```

Set a compare value for a PWM counter.

**Parameters**

- **channel** The channel to use
- **value**  The value to toggle on

**Returns**
On success a 0, otherwise -1

### 2.18.3.7 pwm_levels

```c
int8_t pwm_levels ( uint8_t channel,
                      pwm_state_t initstate,
                      pwm_restore_t restorestate
                  )
```

Set up the logic levels for a PWM counter.

**Parameters**

- **channel** The channel to use
- **initstate**  The initial state of the counter (high or low)
- **restorestate** The rollover restore setting

**Returns**
On success a 0, otherwise -1

### 2.18.3.8 pwm_trigger

```c
int8_t pwm_trigger ( pwm_trigger_t trigger )
```

Set the external trigger settings.

**Parameters**
trigger The trigger setting

**Returns**

On success a 0, otherwise -1
2.19 PWM Audio

The file `ft900_pwm_pcm.h` contains the definitions for the PWM audio functions in the libft900.a library.

2.19.1 API Cross Reference

It utilises the following library APIs:
- `ft900_pwm.h` – Pulse Width Modulation
- `ft900_asm.h` – FT90X and FT93X assembler definitions

Additional definitions are taken from:
- `ft900_registers.h` – FT90X and FT93X register definitions

2.19.2 Enumeration Type Documentation

2.19.2.1 `pwm_pcm_channels_t`

```c
enum pwm_pcm_channels_t
{
    pwm_pcm_channels_mono, Mono
    pwm_pcm_channels_stereo, Stereo
};
```

PWM Channel selection.

2.19.2.2 `pwm_pcm_data_size_t`

```c
enum pwm_pcm_data_size_t
{
    pwm_pcm_data_size_8, 8 bit
    pwm_pcm_data_size_16, 16 bit
};
```

PWM data size selection.

2.19.2.3 `pwm_pcm_endianness_t`

```c
enum pwm_pcm_endianness_t
{
    pwm_pcm_endianness_big, Big endian data
    pwm_pcm_endianness_little, Little endian data
};
```

PWM endianness selection.

2.19.2.4 `pwm_pcm_filter_t`

```c
enum pwm_pcm_filter_t
{

};
```
PWM PCM Filter.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_pcm_filter_off</td>
<td>Off</td>
</tr>
<tr>
<td>pwm_pcm_filter_on</td>
<td>On</td>
</tr>
</tbody>
</table>

### 2.19.2.5 pwm_pcm_interrupt_t

enum pwm_pcm_interrupt_t

PWM PCM Interrupt selection.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_pcm_interrupt_empty</td>
<td>FIFO Empty</td>
</tr>
<tr>
<td>pwm_pcm_interrupt_full</td>
<td>FIFO Full</td>
</tr>
<tr>
<td>pwm_pcm_interrupt_half_full</td>
<td>FIFO Half Full</td>
</tr>
<tr>
<td>pwm_pcm_interrupt_overflow</td>
<td>FIFO Overflow</td>
</tr>
<tr>
<td>pwm_pcm_interrupt_underflow</td>
<td>FIFO Underflow</td>
</tr>
</tbody>
</table>

### 2.19.2.6 pwm_pcm_volume_t

enum pwm_pcm_volume_t

PWM PCM Volume.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_pcm_volume_mute</td>
<td></td>
</tr>
<tr>
<td>pwm_pcm_volume_6</td>
<td>6.25% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_12</td>
<td>12.5% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_19</td>
<td>19% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_25</td>
<td>25% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_31</td>
<td>31% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_37</td>
<td>37% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_44</td>
<td>44% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_50</td>
<td>50% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_56</td>
<td>56% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_63</td>
<td>63% volume</td>
</tr>
<tr>
<td>pwm_pcm_volume_69</td>
<td>69% volume</td>
</tr>
</tbody>
</table>
### 2.19.3 Function Documentation

#### 2.19.3.1 `pwm_pcm_open`

```c
int8_t pwm_pcm_open ( pwm_pcm_channels_t channels,
                       uint16_t samplerate,
                       pwm_pcm_data_size_t datasize,
                       pwm_pcm_endianness_t endianness,
                       pwm_pcm_filter_t filter )
```

Initialise the PWM PCM output.

**Parameters**
- `channels`: The number of channels to output.
- `samplerate`: The sample rate of the audio data.
- `datasize`: The word size of the samples (8 or 16 bit).
- `endianness`: The endianness of the 16 bit word. In 8 bit mode this will be ignored.
- `filter`: If a PCM filter will be used to filter out the PWM carrier.

**Returns**
On success a 0, otherwise -1

#### 2.19.3.2 `pwm_pcm_close`

```c
int8_t pwm_pcm_close ( void )
```

Close the PWM PCM Output.

**Returns**
On success a 0, otherwise -1

#### 2.19.3.3 `pwm_pcm_volume`

```c
int8_t pwm_pcm_volume ( pwm_pcm_volume_t vol )
```

Set the volume of the PWM PCM device.

**Parameters**
- `vol`: The volume to set to.
Returns
On success a 0, otherwise -1

2.19.3.4 pwm_pcm_write

int8_t pwm_pcm_write ( uint16_t data )
Write a word of data to the PWM PCM device.
Parameters
    data The data to write. If 8 bit mode is selected, the top 8 bits will be ignored

Returns
The number of bytes written to the FIFO, otherwise -1

2.19.3.5 pwm_pcm_writen

int8_t pwm_pcm_writen ( uint16_t * data, size_t len )
Write a number of words of data to the PWM PCM device.
Parameters
    data The data to write. If 8 bit mode is selected, the top 8 bits will be ignored
    len  The size of data to write.

Returns

2.19.3.6 The number of bytes written to the FIFO, otherwise -1
pwm_pcm_disable_interrupt

int8_t pwm_pcm_disable_interrupt ( pwm_pcm_interrupt_t interrupt )
Disable an interrupt.
Parameters
    interrupt The interrupt to disable

Returns
On success a 0, otherwise -1

2.19.3.7 pwm_pcm_enable_interrupt

int8_t pwm_pcm_enable_interrupt ( pwm_pcm_interrupt_t interrupt )
Enable an interrupt.
Parameters
    interrupt The interrupt to enable

Returns
On success a 0, otherwise -1
2.19.3.8  \texttt{pwm_pcm_is_interrupted}

\begin{verbatim}
inta_pwm_pcm_is_interrupted ( ppcm_pwm_interrupt_t interrupt )
\end{verbatim}

Query if an interrupt has fired.

**Parameters**

- \texttt{interrupt} The interrupt to query

**Warning**

This function will clear the interrupt being queried and the global PWM interrupt flag

**Returns**

1 for if PWM is interrupted, 0 if PWM is not interrupted, -1 otherwise
2.20 Real Time Clock

The file `ft900_rtc.h` contains the definitions for the real time clock functions in the libft900.a and libft930.a library. The RTC API is meant for the on-chip RTC on FT90X Revision C and FT93X which are the same. However the RTC API is also made backward compatible with FT90X Revision B, where the RTC hardware block is different from that of FT93X/FT90X rev C. The definitions below indicate wherever there is a difference for FT90X revision B.

2.20.1 FT90X and FT93X register definitions API Cross Reference

Additional definitions are taken from:
- `ft900_registers.h` – FT90X and FT93X register definitions
- `time.h` – C library time.h, used to access `struct tm`

2.20.2 Enumeration Type Documentation

2.20.2.1 `rtc_interrupt_t`

`enum rtc_interrupt_t`

RTC Interrupts

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtc_interrupt_alarm1</td>
<td>Alarm 1 Interrupt</td>
</tr>
<tr>
<td>rtc_interrupt_alarm2</td>
<td>Alarm 2 Interrupt</td>
</tr>
<tr>
<td>rtc_interrupt_ext_osc_stopped</td>
<td>External Oscillator Stopped Interrupt</td>
</tr>
<tr>
<td>rtc_interrupt_int_osc_stopped</td>
<td>Internal Oscillator Stopped Interrupt</td>
</tr>
<tr>
<td>rtc_interrupt_int_auto_calibration</td>
<td>Auto Calibration Interrupt</td>
</tr>
<tr>
<td>rtc_interrupt_max</td>
<td>Placeholder for maximum value</td>
</tr>
<tr>
<td>rtc_legacy_interrupt_alarm</td>
<td>This ENUM is ONLY applicable to FT90x Revision B on-chip RTC and is the only enum to be used.</td>
</tr>
</tbody>
</table>

2.20.2.2 `rtc_alarm_type_t`

`enum rtc_alarm_type_t`

RTC Alarm Types

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtc_alarm_1Hz</td>
<td>Alarm every second</td>
</tr>
<tr>
<td>rtc_alarm_match_sec</td>
<td>Alarm when seconds match</td>
</tr>
<tr>
<td>rtc_alarm_match_min_sec</td>
<td>Alarm when seconds and minutes match</td>
</tr>
<tr>
<td>rtc_alarm_match_hr_min_sec</td>
<td>Alarm when seconds, minutes and hours match</td>
</tr>
<tr>
<td>rtc_alarm_match_date_hr_min_sec</td>
<td>Alarm when seconds, minutes, hours and date matches</td>
</tr>
</tbody>
</table>
2.20.2.3 rtc_option_t

enum rtc_option_t
RTC Option

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtc_option_auto_refresh</td>
<td>Auto Refresh Option</td>
</tr>
<tr>
<td>rtc_option_wrap</td>
<td>Enable or disable RTC Wrap around (Applicable ONLY in FT900 Revision B)</td>
</tr>
<tr>
<td>rtc_option_mask_interrupt</td>
<td>Option to set whether the Real Time Clock module should mask the match interrupt line (Applicable ONLY in FT90X Revision B)</td>
</tr>
</tbody>
</table>

2.20.3 Function Documentation

2.20.3.1 rtc_init

int8_t rtc_init ( void )
Initialise the Real Time Clock.

Returns
0 on success, -1 otherwise

2.20.3.2 rtc_start

int8_t rtc_start ( void )
Start the Real Time Clock.

Returns
0 on success, -1 otherwise

2.20.3.3 rtc_stop

int8_t rtc_stop ( void )
Stop the Real Time Clock.

Returns
0 on success, -1 otherwise

2.20.3.4 rtc_read

int8_t rtc_read ( struct tm* const time )
Read the current value of the Real Time Clock.
Parameters

*time* A pointer of type `struct tm*` (defined in `<time.h>`) to which the RTC time is to be read in case of FT93X and FT90X rev C. Only the field `tm.sec` is written to with current RTC time read, in case of FT90X rev B.

Returns

0 on success, -1 otherwise

### 2.20.3.5 `rtc_write`

```c
int8_t rtc_write ( const struct tm* time )
```

Write the given date/time to the Real Time Clock

**Parameters**

*time* A pointer of type `struct tm*` (defined in `<time.h>`) contains the time to be written into the RTC for FT93X and FT90X rev C. Only the field `tm.sec` should contain the RTC time to be written in case of FT90X rev B.

Returns

0 on success, -1 otherwise

### 2.20.3.6 `rtc_option`

```c
int8_t rtc_option ( rtc_option_t opt, uint8_t val )
```

Control options of the RTC

**Parameters**

*opt* The type of option to update

*val* The value of the selected option

Returns

0 on success, -1 otherwise

### 2.20.3.7 `rtc_set_alarm`

```c
int8_t rtc_set_alarm ( uint8_t number,
                        struct tm* time,
                        rtc_alarm_type alarm_type )
```

Set an Alarm on the RTC

**Parameters**

*number* The number of the Alarm to set (1 or 2)

*time* The date/time of the Alarm

*alarm_type* The type of alarm match required

Returns
0 on success, -1 otherwise

2.20.3.8 rtc_is_interrupted

int8_t rtc_is_interrupted ( rtc_interrupt_t Interrupt)
Check if an interrupt has been triggered.

Parameters

interrupt The interrupt to check

Warning
This function clears the current interrupt status bit

Returns
1 when interrupted, 0 when not interrupted, -1 otherwise

2.20.3.9 rtc_enable_interrupt

int8_t rtc_enable_interrupt ( rtc_interrupt_t Interrupt)
Enable an interrupt on the RTC.

Parameters

interrupt The interrupt to enable

Returns
0 on success, -1 otherwise

2.20.3.10 rtc_enable_interrupts_globally

int8_t rtc_enable_interrupts_globally ( void )
Enable RTC interrupts

Returns
0 on success, -1 otherwise

2.20.3.11 rtc_disable_interrupt

int8_t rtc_disable_interrupt ( rtc_interrupt_t Interrupt)
Disable an interrupt on the RTC.

Parameters

interrupt The interrupt to disable

Returns
0 on success, -1 otherwise

2.20.3.12 rtc_disable_interrupts_globally

int8_t rtc_disable_interrupts_globally ( void )
Disable the RTC from interrupting.

Returns
0 on success, -1 otherwise
2.21 USB Device Stack API

The file `ft900_usbd.h` contains the definitions for the USB device functions in the `libft900.a` and `libft930.a` library. This contains USB Device API function definitions, constants and structures which are exposed in the API.

Note that as this is a USB device all transaction nomenclature is from the point of view from the host. If the device sends data to the host then it is called an IN transaction, if it receives data from the host then it is an OUT transaction.

**Synchronous transfer and Asynchronous transfer:**

The `ft900_usbd.h` provides synchronous transfer function i.e., the caller waits until the return of the callee, which is the end of the transfer to the USB endpoint.

Using APIs in `ft900_usbd.h`,

- Application prepares and processes the payload buffer
- Application calls `USBD_transfer()` to transfer the buffer
  - `USBD_Transfer` is a synchronous call that waits for EP ready and sends the data in batches of 512 and return.
- Application processes the buffer again only after `USBD_transfer` returns.

There are extended APIs defined in `ft900_usbdx.h` that provides asynchronous transfer of USB data.

- USB manages buffer, buffer is divided into chunks called USB Request Blocks (URBs) and are marked with ownership as ‘USBD owned’ or ‘Application owned’.
- USB will transfer its owned URB to the USBD ISR, and releases the empty URB(s) back to application (change ownership)
- Application can
  - get owned URB(s) in main loop or another ISR
  - process URB(s)
  - queue back URB(s) to USBD (change ownership)

More information about USDB Device Stack Extension API is found in the [next section](#).

### 2.21.1 API Cross Reference

Utilises the following library APIs:

- `ft900_gpio.h` – General Purpose I/O and Pad Control
- `ft900_sys.h` – Chip Management
- `ft900_delay.h` – Delay
- `ft900_interrupt.h` – Interrupt Management

Additional definitions are taken from:

- `ft900_usb.h` – General USB definitions
- `ft900_registers.h` – FT90X and FT93X register definitions
2.21.2 Macro Definition Documentation

2.21.2.1 USB Device Error Codes

#define USBD_ERR_DISCONNECTED -10
Device not configured by host.
Device physically disconnected from host.
#define USBD_ERR_INCOMPLETE -5
Incomplete/interrupted transfer.
#define USBD_ERR_RESOURCES -4
Not enough endpoint resources.
#define USBD_ERR_NOT_SUPPORTED -3
Operation not supported.
#define USBD_ERR_NOT_CONFIGURED -2
Endpoint not configured.
#define USBD_ERR_INVALID_PARAMETER -1
Invalid parameter supplied to API function.

2.21.3 Typedef Documentation

2.21.3.1 USBD_reset_callback

typedef void(* USBD_reset_callback) (uint8_t status)
Callback declaration for a host reset.

Parameters

[in] status Unused.

2.21.3.2 USBD_suspend_callback

typedef void(* USBD_suspend_callback) (uint8_t status)
Callback declaration for a suspend/resume.

Parameters

[in] status Unused.

2.21.3.3 USBD_request_callback

typedef int8_t(* USBD_request_callback) (USB_device_request *req)
Callback declaration for Vendor, Class and optionally Standard USB requests.

Parameters

[in] req USB request.
**Returns**

USBD_OK if the request was handled successfully; any other return value (such as FT9XX_FAILED) causes the USB driver to stall the control endpoints.

### 2.21.3.4 USBD_descriptor_callback

typedef int8_t (*USBD_descriptor_callback) (USB_device_request *req, uint8_t **buffer, uint16_t *len)

Callback declaration for standard get descriptor requests to obtain descriptor data.

**Parameters**

- **[in]** req  USB request.
- **[in]** buffer Data buffer containing descriptor.

**Returns**

USBD_OK if the request was handled successfully; any other return value (such as FT9XX_FAILED) causes the USB driver to stall the control endpoints.

### 2.21.3.5 USBD_ep_callback

typedef void (*)(USBD_ENDPOINT_NUMBER ep_number)

Callback declaration for transaction completion on endpoint. The endpoint number is passed to the callback to allow the same function to handle multiple endpoints.

### 2.21.4 Enumeration Type Documentation

#### 2.21.4.1 USBD_STATE

enum USBD_STATE

USB States. USB Spec section 9.1.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_STATE_NONE</td>
<td>Device is not attached.</td>
</tr>
<tr>
<td>USBD_STATE_ATTACHED</td>
<td>Device is attached to USB.</td>
</tr>
<tr>
<td>USBD_STATEPOWERED</td>
<td>Device is attached and powered.</td>
</tr>
<tr>
<td>USBD_STATE_DEFAULT</td>
<td>Device is attached, has power and has been reset.</td>
</tr>
<tr>
<td>USBD_STATE_ADDRESS</td>
<td>Unique device address has not been set.</td>
</tr>
<tr>
<td>USBD_STATE_CONFIGURED</td>
<td>Unique device address is now assigned. Device can be used by host.</td>
</tr>
<tr>
<td>USBD_STATE_SUSPENDED</td>
<td>Device has been suspended.</td>
</tr>
</tbody>
</table>

#### 2.21.4.2 USBD DEVICE_SPEED

enum USBD_DEVICE_SPEED

USB Endpoint Speed setting.


<table>
<thead>
<tr>
<th>USBD_SPEED_FULL</th>
<th>Full speed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_SPEED_HIGH</td>
<td>High speed.</td>
</tr>
</tbody>
</table>

### 2.21.4.3 USBD_ENDPOINT_NUMBER

```c
enum USBD_ENDPOINT_NUMBER
```

USB Endpoint Numbers.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_EP_0</td>
<td>Endpoint 0 (Control Endpoint)</td>
</tr>
<tr>
<td>USBD_EP_1</td>
<td>Endpoint 1.</td>
</tr>
<tr>
<td>USBD_EP_2</td>
<td>Endpoint 2.</td>
</tr>
<tr>
<td>USBD_EP_3</td>
<td>Endpoint 3.</td>
</tr>
<tr>
<td>USBD_EP_5</td>
<td>Endpoint 5.</td>
</tr>
<tr>
<td>USBD_EP_7</td>
<td>Endpoint 7.</td>
</tr>
<tr>
<td>USBD_EP_8</td>
<td>Endpoint 8. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_9</td>
<td>Endpoint 9. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_10</td>
<td>Endpoint 10. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_11</td>
<td>Endpoint 11. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_12</td>
<td>Endpoint 12. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_13</td>
<td>Endpoint 13. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_14</td>
<td>Endpoint 14. [FT93X only]</td>
</tr>
<tr>
<td>USBD_EP_15</td>
<td>Endpoint 15. [FT93X only]</td>
</tr>
</tbody>
</table>

### 2.21.4.4 USBD_ENDPOINT_DIR

```c
enum USBD_ENDPOINT_DIR
```

USB Endpoint Direction.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_DIR_OUT</td>
<td>Direction host to device.</td>
</tr>
<tr>
<td>USBD_DIR_IN</td>
<td>Direction device to host.</td>
</tr>
</tbody>
</table>
2.21.4.5 USBD_ENDPOINT_SIZE

enum USBD_ENDPOINT_SIZE

USB Endpoint Sizes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_EP_SIZE_8</td>
<td>8 Bytes</td>
</tr>
<tr>
<td>USBD_EP_SIZE_16</td>
<td>16 Bytes</td>
</tr>
<tr>
<td>USBD_EP_SIZE_32</td>
<td>32 Bytes</td>
</tr>
<tr>
<td>USBD_EP_SIZE_64</td>
<td>64 Bytes</td>
</tr>
</tbody>
</table>

2.21.4.6 USBD_ENDPOINT_TYPE

enum USBD_ENDPOINT_TYPE

USB Endpoint Types.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_EP_TYPE_DISABLED</td>
<td>Disabled.</td>
</tr>
<tr>
<td>USBD_EP_BULK</td>
<td>Bulk Endpoint.</td>
</tr>
<tr>
<td>USBD_EP_INT</td>
<td>Interrupt Endpoint.</td>
</tr>
<tr>
<td>USBD_EP_CTRL</td>
<td>Control Endpoint.</td>
</tr>
</tbody>
</table>

2.21.4.7 USBD_ENDPOINT_DB

enum USBD_ENDPOINT_DB

USB Endpoint Double Buffering Enable.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_DB_OFF</td>
<td>Disabled.</td>
</tr>
<tr>
<td>USBD_DB_ON</td>
<td>Enabled.</td>
</tr>
</tbody>
</table>
### 2.21.4.8 USBD_TESTMODE_SELECT

```
enum USBD_TESTMODE_SELECT
```

Enums used to select the test modes. For more information refer to Section 7.1.20 of USB2.0 Specification.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_TEST_J</td>
<td>Test mode Test_J</td>
</tr>
<tr>
<td>USBD_TEST_K</td>
<td>Test mode Test_K</td>
</tr>
<tr>
<td>USBD_TEST_SE0_NAK</td>
<td>Test mode Test_SE0_NAK</td>
</tr>
<tr>
<td>USBD_TEST_PACKET</td>
<td>Test mode Test_PACKET</td>
</tr>
</tbody>
</table>

### 2.21.5 Structure Documentation

#### 2.21.5.1 USBD_ctx

Struct containing callback functions for the USB upper layer driver and callback functions for USB suspend/resume and USB reset. Sets USBD configuration information.

**Data Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_descriptor_callback</td>
<td>get_descriptor_cb</td>
</tr>
<tr>
<td>USBD_set_configuration_callback</td>
<td>set_configuration_cb</td>
</tr>
<tr>
<td>USBD_request_callback</td>
<td>standard_reqCb</td>
</tr>
<tr>
<td>USBD_set_interface_callback</td>
<td>set_interfaceCb</td>
</tr>
<tr>
<td>USBD_get_interface_callback</td>
<td>get_interfaceCb</td>
</tr>
<tr>
<td>USBD_request_callback</td>
<td>class_reqCb</td>
</tr>
<tr>
<td>USBD_request_callback</td>
<td>vendor_reqCb</td>
</tr>
<tr>
<td>USBD_request_callback</td>
<td>ep_feature_reqCb</td>
</tr>
<tr>
<td>USBD_request_callback</td>
<td>feature_reqCb</td>
</tr>
<tr>
<td>USBD_suspend_callback</td>
<td>suspendCb</td>
</tr>
<tr>
<td>USBD_suspend_callback</td>
<td>resumeCb</td>
</tr>
<tr>
<td>USBD_suspend_callback</td>
<td>lpmCb</td>
</tr>
<tr>
<td>USBD_reset_callback</td>
<td>resetCb</td>
</tr>
<tr>
<td>USBD_suspend_callback</td>
<td>sofCb</td>
</tr>
<tr>
<td>USBD_ENDPOINT_SIZE</td>
<td>ep0_size</td>
</tr>
<tr>
<td>USBD_ep_callback</td>
<td>ep0Cb</td>
</tr>
<tr>
<td>USBD_DEVICE_SPEED</td>
<td>speed</td>
</tr>
<tr>
<td>uint8_t</td>
<td>lowPwrSuspend</td>
</tr>
</tbody>
</table>

**Field Documentation**
class_req_cb
[Optional] class request callback function.

ep0_cb
Callback function for a data transfer to or from the control endpoint.

ep0_size
Endpoint size for control endpoints. Section 3.3.1 FT900 USB Program Manual. Sets DC_EP0_CONTROL register in Table 3.6.
0: 8 bytes. 1: 16 bytes. 2: 32 bytes. 3: 64 bytes.

get_descriptor_cb
[Optional] Descriptor callback function.
Handler function to obtain descriptors (device, configuration, string, HID, Hub etc.) for use with the built-in USB standard request handler. This must be present if the standard request handler callback is not used.

set_configuration_cb
[Optional] Handler function to check set configuration is valid for application. For use with the built-in USB standard request handler. If this is not present then the default handling of the request will occur.

set_interface_cb
[Optional] Handler function to set the alternate settings for an interface. For use with the built-in USB standard request handler. If this is not present then the request will be stalled.

get_interface_cb
[Optional] Handler function to return the alternate settings for an interface. For use with the built-in USB standard request handler. If this is not present then the request will be stalled.

ep_feature_req_cb
[Optional] Endpoint Feature request callback function.

feature_req_cb
[Optional] Device Feature (Remote Wakeup) request callback function.

lowPwrSuspend
Device power control. Engage power saving mode if bus-powered and suspend state entered.
0: Disable. 1: Enabled.

lpm_cb
[Optional] USB bus LPM (Link Power Management) callback function.

reset_cb
[Optional] USB bus reset callback function.

resume_cb
[Optional] USB bus suspend callback function.
sof_cb

[Optional] USB SOF callback function.

speed

Device configuration section. High speed/full speed select. Section 3.2.2 FT900 USB Program Manual.

0: Full speed only. 1: High speed if available.

standard_req_cb


Handler for USB standard requests. This is used for overriding the built-in standard request handler to customise the responses to standard requests. If it is not set then the built-in handler will be used and the descriptor_cb function used to obtain descriptors.

suspend_cb

[Optional] USB bus suspend callback function.

vendor_req_cb

[Optional] vendor request callback function.

### 2.21.6 Function Documentation

#### 2.21.6.1 USBD initialise

```c
void USBD_initialise ( USBD_ctx * ctx )
```

Initialise USB hardware.

Performs a software reset and initialises the USB hardware. The `USBD_ctx` contains function pointers to the protocol layer handling USB requests. Appropriate USB requests will be routed to the correct handler, whether that is Standard, Class or Vendor requests. A device may not need a handler for Vendor or Class requests depending on the device configuration.

Optional function pointers are also available for USB suspend and resume call-backs and bus resets issued by the host.

This function MUST be called prior to any further call to the USB functions.

**Parameters**

- **[in]** `ctx` USB context.

#### 2.21.6.2 USBD finalise

```c
void USBD_finalise ( void )
```

Finalise USB hardware.

Releases any resources associated with the USB driver and disables the hardware.

#### 2.21.6.3 USBD attach

```c
void USBD_attach ( void )
```

Attach USB hardware.

Attaches the USB device to the USB host after a USBD_detach call.
2.21.6.4 USBD_detach

void USBD_detach ( void )

Detach USB hardware.
Detaches the USB device from the USB host. This will look like a device disconnect to the host and it will act like the device is removed.

2.21.6.5 USBD_is_connected

int8_t USBD_is_connected ( )

Check if the device is connected to a host (or external power source).
Checks the VBUS detect line for a host connected.

2.21.6.6 USBD_set_state

void USBD_set_state ( USBD_STATE state )

Set USB state.
Sets the current state of the current USB device. Please refer to section 9.1 of the USB 2.0 spec for more information.

Parameters

[in] state The new state of the current USB device.

2.21.6.7 USBD_create_endpoint

int8_t USBD_create_endpoint ( USBD_ENDPOINT_NUMBER ep_number,
                             USBD_ENDPOINT_TYPE ep_type,
                             USBD_ENDPOINT_DIR ep_dir,
                             USBD_ENDPOINT_SIZE ep_size,
                             USBD_ENDPOINT_DB ep_db,
                             USBD_ep_callback ep_cb )

Create a USB endpoint.
Creates an endpoint with the requested properties.

There is a total of 4 kB of RAM for FT90x Rev B, 6kB for FT90x Rev C and 8kB for FT930. This total RAM is for all the IN and OUT endpoints. Therefore the total max packet for all IN endpoints and OUT endpoints must be less than this figure. If double buffering is employed for an endpoint then it will use twice the amount of RAM. .

Parameters

[in] ep_number USB endpoint number. (N/A for control endpoints).
[in] ep_type USB endpoint type: BULK, ISO or INT. (N/A for control endpoints).
[in] ep_dir Endpoint direction, In or Out.
[in] ep_size USB endpoint max packet size in bytes.
[in] ep_db USB endpoint double buffering enable. (N/A for control endpoints).
[in] ep_cb Callback functions for this endpoint. This function will be called from the USBD_process function when an event concerned with the endpoint has occurred. This can be used for receiving notification of a transaction to or from the endpoint heralding the availability of data (OUT endpoints) or the completion of a transmission of data (IN endpoints). However, the USBD_ep_buffer_full() function can be polled to determine the same status if callbacks are inappropriate.

Returns
USBD_OK if successful.
USBD_ERR_NOT_SUPPORTED if an endpoint higher than the maximum number of endpoints is requested.
USBD_ERR_INVALID_PARAMETER if an illegal endpoint size is requested.
USBD_ERR_RESOURCES if there is not enough endpoint RAM for the endpoint size requested.

2.21.6.8 USBD_free_endpoint

int8_t USBD_free_endpoint ( USBD_ENDPOINT_NUMBER ep_number )
Free USB endpoint.
Disable and free the specified endpoint.

Parameters
[in] ep USB endpoint handle.

Returns
USBD_OK if successful
USBD_ERR_NOT_CONFIGURED if endpoint is not configured.
USBD_ERR_INVALID_PARAMETER if endpoint number not allowed.

2.21.6.9 USBD_get_bus_speed

USBD_DEVICE_SPEED USBD_get_bus_speed ( void )
To be called to get the current USB bus speed at which USBD operates

Returns
Full or High Speed.

2.21.6.10 USBD_connect

int8_t USBD_connect ( void )
Connect to a USB host.
Checks the VBUS detect line for a host connected and proceed to allow the device to negotiate a connection to the host.

Returns
USBD_OK on success.
USBD_ERR_INVALID_PARAMETER if req is invalid.

2.21.6.11 USBD_timer

void USBD_timer ( void )
USB timer.
To be called every millisecond from an interrupt handler to provide timeout support for USB device transactions. This will check all pending transfers, decrement timeout values and expire any timed out transactions.

2.21.6.12 **USBD_process**

```c
int8_t USBD_process ( void )
```

USB process.

To be continuously called by the user application or USB device thread. Checks for control endpoint transfer activity and invoke relevant callback. Manages suspend and resume states and power management.

**Note:** This function is deprecated.

**Returns**

Non-zero if USB transaction has been processed.

2.21.6.13 **USBD_transfer**

```c
int32_t USBD_transfer ( USBD_ENDPOINT_NUMBER ep_number,
                        uint8_t * buffer,
                        size_t _t length
                        )
```

Transfer data to/from a non-control USB endpoint.

USB IN or OUT request is implied from the settings of the endpoint passed as a parameter.

**Parameters**

- **[in]** `ep_number` USB endpoint number.
- **[in]** `buffer` Appropriately sized buffer for the transfer.
- **[in]** `length` For IN transfers, the number of bytes to be sent. For OUT transfers, the maximum number of bytes to be read.

**Returns**

The number of bytes actually transferred.

- `USBD_ERR_NOT_CONFIGURED` if endpoint is not configured.
- `USBD_ERR_INVALID_PARAMETER` if endpoint number not allowed.

2.21.6.14 **USBD_transfer_ex**

```c
int32_t USBD_transfer_ex ( USBD_ENDPOINT_NUMBER ep_number,
                          uint8_t * buffer,
                          size_t _t length,
                          int8_t _t part,
                          size_t _t offset
                          )
```

Transfer data to/from a non-control USB endpoint with options.

USB IN or OUT request is implied from the settings of the endpoint passed as a parameter. The end-of-packet will not be sent when the data from the buffer parameter is sent.
This will allow a follow-on USBD_transfer_ex call to either send more data (with the part parameter non-zero and a correct offset set) or an end-of-packet with part not set. This allows a USB data packet to a non-control endpoint to be formed from multiple calls with data from potentially different places.

**Parameters**

- **[in]** `ep_number` USB endpoint number.
- **[in]** `buffer` Appropriately sized buffer for the transfer.
- **[in]** `length` For IN transfers, the number of bytes to be sent. For OUT transfers, the maximum number of bytes to read.
- **[in]** `part` Signifies that this is a partial transfer.
- **[in]** `offset` Offset (within the current packet) from where to continue for subsequent calls when using partial packets.

**Returns**
The number of bytes actually transferred.
USBD_ERR_NOT_CONFIGURED if endpoint is not configured.
USBD_ERR_INVALID_PARAMETER if endpoint number not allowed.

### 2.21.6.15 **USBD_transfer_ep0**

```c
int32_t USBD_transfer_ep0 ( USBD_ENDPOINT_DIR dir,
                          uint8_t * buffer,
                          size_t dataLength,
                          size_t requestLength
)
```

Transfer data to/from a USB control endpoint. Endpoint number is assumed to be zero.

**Parameters**

- **[in]** `dir` Control endpoint data direction.
- **[in]** `buffer` Appropriately sized buffer for the transfer.
- **[in]** `dataLength` For IN transfers, the number of bytes to be sent. For OUT transfers, the maximum number of bytes to read.
- **[in]** `requestLength` The number of bytes requested by the host in the wLength field of the SETUP packet.

**Returns**
The number of bytes actually transferred.
USBD_ERR_NOT_CONFIGURED if endpoint is not configured.

### 2.21.6.16 **USBD_clear_endpoint**

```c
int8_t USBD_clear_endpoint ( USBD_ENDPOINT_NUMBER ep_number )
```

Clears endpoint stall.

Clears a stall from the specified endpoint. The default standard request handler will call this function for a CLEAR_FEATURE endpoint request.
Parameters

[in] ep_number USB endpoint number.

Returns

USBD_OK if successful
USBD_ERR_NOT_CONFIGURED if endpoint is not configured.
USBD_ERR_INVALID_PARAMETER if endpoint number not allowed.

2.21.6.17 USBD_ep_max_size

uint16_t USBD_ep_max_size ( USBD_ENDPOINT_NUMBER ep_number )

Find Max Packet Size of USB endpoint.

Parameters

[in] ep_number USB endpoint number.

Returns

Return the maximum number of bytes which can be sent or received single USB packets for an endpoint.
USBD_ERR_NOT_CONFIGURED if the endpoint is not configured.
USBD_ERR_INVALID_PARAMETER if the endpoint number is not allowed.

2.21.6.18 USBD_ep_buffer_full

int8_t USBD_ep_buffer_full ( USBD_ENDPOINT_NUMBER ep_number )

Get USB endpoint buffer status.

Returns the current buffer status of an endpoint using the SELECT_ENDPOINT call.

Parameters

[in] ep_number USB endpoint number.

Returns

Current state of the endpoint buffer. TRUE if full, FALSE if empty.

2.21.6.19 USBD_get_ep_stalled

int8_t USBD_get_ep_stalled ( USBD_ENDPOINT_NUMBER ep_number )

Get USB endpoint stall status.

Returns the current stall status of an endpoint using the SELECT_ENDPOINT call.

Parameters

[in] ep_number USB endpoint number.

Returns

Current stall state of the endpoint.
>0 if stalled, zero if not stalled.
USBD_ERR_NOT_CONFIGURED if the endpoint is not configured.
USBD_ERR_INVALID_PARAMETER if the endpoint number is not allowed.

2.21.6.20 USBD_stall_endpoint

int8_t USBD_stall_endpoint ( USBD_ENDPOINT_NUMBER ep_number )
Stall endpoint.
Stalls the specified endpoint. The default standard request handler will call this function for a
SET_FEATURE endpoint request.

**Parameters**

[in] `ep_number` USB endpoint number.

**Returns**

USBD_OK if successful
USBD_ERR_NOT_CONFIGURED if the endpoint is not configured.
USBD_ERR_INVALID_PARAMETER if the endpoint number is not allowed.

### 2.21.6.21 USBD_get_state

USBD_STATE USBD_get_state ( void )

Get USB state.

Returns the current state of the current USB device. Please refer to section 9.1 of the USB 2.0
spec for more information.

**Returns**

Current state of the current USB device.

### 2.21.6.22 USBD_req_get_configuration

`int8_t USBD_req_get_configuration ( void )`

Handles GET_CONFIGURATION request.

Handles the DATA phase of a GET_CONFIGURATION request from the host. The application has to
respond with SETUP ACK or STALL. The default standard request handler will call this function; if
the handler is overridden then the application must call this when this request is received.

**Returns**

USBD_OK on success.
USBD_ERR_INVALID_PARAMETER if req is invalid.

### 2.21.6.23 USBD_req_set_address

`int8_t USBD_req_set_address ( USB_device_request * req )`

Handles SET_ADDRESS request.

Places the device in the ADDRESS state. The application has to respond with SETUP ACK or STALL.
The default standard request handler will call this function; if the handler is overridden then the
application must call this when this request is received.

**Parameters**

[in] `req` USB request.

**Returns**

USBD_OK on success.
USBD_ERR_INVALID_PARAMETER if req is invalid.

### 2.21.6.24 USBD_req_set_configuration

`int8_t USBD_req_set_configuration ( USB_device_request * req )`

Handles SET_CONFIGURATION request.
Places the device in the CONFIGURED state or puts it back into the ADDRESS state. The application has to respond with SETUP ACK or STALL. The default standard request handler will call this function; if the handler is overridden then the application must call this when this request is received.

**Parameters**

- [in] `req` USB request.

**Returns**

- `USBD_OK` on success.
- `USBD_ERR_INVALID_PARAMETER` if `req` is invalid.

#### 2.21.6.25 USBD_get_remote_wakeup

```c
uint8_t USBD_get_remote_wakeup ( void )
```

Get USB remote wakeup feature status.

Returns the current feature status of remote wakeup.

**Returns**

Current remote wakeup feature status. TRUE if enabled, FALSE if not enabled.

#### 2.21.6.26 USBD_set_remote_wakeup

```c
void USBD_set_remote_wakeup ( void )
```

Set USB remote wakeup feature status.

#### 2.21.6.27 USBD_clear_remote_wakeup

```c
void USBD_clear_remote_wakeup ( void )
```

Clear USB remote wakeup feature status.

#### 2.21.6.28 USBD_wakeup

```c
void USBD_wakeup ( void )
```

Drive resume signalling upstream when remote wakeup is enabled.

#### 2.21.6.29 USBD_resume

```c
void USBD_resume ( void )
```

When USB related events like host resume and host reset are detected, PM irq will be received if it is enabled. The firmware needs to remove the SUSPEND from the PHY by calling this function.

#### 2.21.6.30 USBD_ep_data_rx_count(USBD_ENDPOINT_NUMBER ep_number)

```c
void USBD_ep_data_rx_count ( USBD_ENDPOINT_NUMBER ep_number )
```

Provides the size of the OUT packet that is yet to be read.

**Parameters**

- [in] `ep_number` USB endpoint number.

**Returns**
The number of bytes actually received.
USB_ERR_NOT_CONFIGURED if endpoint is not configured.

### 2.21.6.31 USBD_set_testmode

void USBD_set_testmode ( USBD_TESTMODE_SELECT test_selector )

To be called to enter into Test Mode.

**Parameters**

- [in] **test_selector** the type of test to be performed in the Test Mode

### 2.21.6.32 USBD_suspend_device

void USBD_suspend_device ( void )

When device is initialized and no bus activity for certain time, this API can be called to put the USB device to suspend. This API takes the USBD state to USBD_STATE_SUSPENDED.

### 2.22 USB Device Stack Extensions API

The file `ft900_usbdx.h` contains the definitions for the USB device extension functions in the `libft900.a` library.

The application has to provide a linear space buffer to the USBD. An array of URB blocks are then initialized by USBD, dividing the linear space into small chunks (512 bytes for HS, 64 bytes for FS). A Pipe is initialized by USBD that points to its own URBs. Each endpoint is represented by a pipe structure. Application can create multiple pipes.

1. The pipe and the URB structures and URB buffers are to be provided by the application. USBD will initialize the structures and manage them.
2. Application need to take care of the buffer alignment which should at least 4-byte aligned.
3. Application need to implement function `USBD_pipe_isr()` to enable Asynchronous data transfer in USBD. Refer to USBD examples for sample implementation of `USBD_pipe_isr()`.

#### 2.22.1 API Cross Reference

It utilises the following library APIs:

- `ft900_memctl.h` – FT9xx memory controller driver

#### 2.22.2 Structure Documentation

##### 2.22.2.1 urb

Singly linked list structure for maintaining URBs out of a linear buffer passed by the application.

**Data Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>uint8_t</td>
<td>Start of URB</td>
</tr>
<tr>
<td>ptr</td>
<td>uint8_t</td>
<td>Reset to start of linear buffer after data transfer. OUT: start of buffer to be processed IN: end of data to be transferred</td>
</tr>
<tr>
<td>end</td>
<td>uint8_t</td>
<td>For OUT endpoint: end of data to be transferred</td>
</tr>
<tr>
<td>owned_by_usbd</td>
<td>bool</td>
<td>For IN endpoint: Maximum packet length</td>
</tr>
<tr>
<td>owned</td>
<td>bool</td>
<td>Ownership bit</td>
</tr>
</tbody>
</table>
uint8_t id  URB number/id
struct urb *next  Pointer to next URB

2.22.2.2 pipe
Structure to maintain the pipe related data. A pipe structure is for each endpoint.

Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct urb</td>
<td>*usbd_urb The address of the start of the URB that is available for USBD to process.</td>
</tr>
<tr>
<td>struct urb</td>
<td>*app_urb The address of the start of the URB that is available for application to process.</td>
</tr>
<tr>
<td>uint8_t</td>
<td>*buf_start Start of the linear buffer</td>
</tr>
<tr>
<td>uint8_t</td>
<td>*buf_end End of the linear buffer</td>
</tr>
<tr>
<td>usbd_callback</td>
<td>on_usbd_ready Application’s callback function registered with USBD will be called at the event when USBD engine is ready.</td>
</tr>
<tr>
<td>usbd_callback</td>
<td>on_usbd_underrun Application’s callback function registered with USBD will be called at the event when USBD engine is underrunning.</td>
</tr>
<tr>
<td>uint8_t</td>
<td>id USBD_ENDPOINT_NUMBER for which a pipe has to be created</td>
</tr>
<tr>
<td>uint8_t</td>
<td>ep USBD_ENDPOINT_NUMBER with MSB bit set in case of IN endpoint</td>
</tr>
<tr>
<td>bool</td>
<td>usbd_paused Set if USBD engine is paused when no application data</td>
</tr>
<tr>
<td>bool</td>
<td>app_paused Application pause itself when no more buffer can be processed, and waiting for USBD to give more data. USBD engine will call the on_usbd_ready() callback once app is paused, to resume the application’s process.</td>
</tr>
</tbody>
</table>

2.22.3 Function Documentation

2.22.3.1 Helper functions
Below list of functions are defined in ft900_usbdx.h to be ‘static inline’ that could become of use for the application to poll the status of the URB.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t urb_get_id(const struct urb *urb)</td>
<td>URB id, for debugging purpose</td>
</tr>
<tr>
<td>uint16_t urb_get_app_to_process(const struct urb *urb)</td>
<td>Length to be processed by application</td>
</tr>
<tr>
<td></td>
<td>IN: length of free space to fill in,</td>
</tr>
<tr>
<td></td>
<td>OUT: length of host data</td>
</tr>
<tr>
<td>uint16_t urb_get_app_consumed(const struct urb *urb)</td>
<td>Length of data already been processed by application</td>
</tr>
<tr>
<td>bool urb_in_fully_filled(const struct urb *urb)</td>
<td>To detect aligned IN data transfer</td>
</tr>
<tr>
<td>bool urb_in_empty(const struct urb *urb)</td>
<td>To detect empty IN URB</td>
</tr>
</tbody>
</table>
bool urb_owned_by_app(const struct urb *urb)  
To detect if application are free to use the URB

### 2.22.3.2 usbd_pipe_init

Initialise the URB pipe for data transfer.

```c
bool usbd_pipe_init(struct pipe *pp, uint8_t id, uint8_t ep, struct urb *urbs, uint8_t *bufs, uint8_t urb_count);
```

Application provides a linear space buffer to USBD. An array of URBS is initialised by USBD, dividing the linear space into small chunks (512 bytes for HS, 64 bytes for FS). A pipe is initialised by USBD, pointing to its own URBS. Each Endpoint is represented by a pipe structure. Application can create multiple pipes using this API.

**Parameters**

- **[in]** `pp` Pipe structure that gets initialised by USBD upon creation of pipe
- **[in]** `id` USBD_ENDPOINT_NUMBER for which a pipe has to be created.
- **[in]** `ep` USBD_ENDPOINT_NUMBER with MSB bit set incase of IN endpoints.
- **[in]** `urbs` URB structure that gets initialised by USBD upon creation of pipe.
- **[in]** `bufs` Linear buffer to be passed by the application which gets divided into URBS.
- **[in]** `urb_count` number of URBS.

**Returns**

'True' if pipe is created. 'False' if the input parameters are invalid.

### 2.22.3.3 usbd_force_acquire_urb_for_app

API to always return a URB.

```c
struct urb *usbd_force_acquire_urb_for_app(struct pipe *pp);
```

Application can choose to acquire URB even if it is still held by USBD.

**Parameters**

- **[in]** `pp` The pipe whose data to be transferred to/from USBD

**Returns**

A URB.

### 2.22.3.4 usbd_submit_urb

API to submit a URB to USBD.

```c
void usbd_submit_urb(struct pipe *pp, struct urb *urb);
```

Application fills the urb and submits to USBD through this API call. When a URB is submitted without filling, this will make USBD to send a ZLP.

**Parameters**

- **[in]** `pp` The pipe whose data to be transferred to/from USBD
- **[in]** `urb` URB filled by application
2.22.3.5 usbd_get_app_urbs

API to be used to get more than one URBs for application.

```c
uint8_t *usbd_get_app_urbs(const struct pipe *pp, uint16_t len)
```

Application shall use this API to get more than one URB from the USBD.

**Note:** This function should be called to obtain the length before copying and submitting the URBs through usbd_submit_urbs().

**Parameters**

- `[in] pp` The pipe whose data to be transferred to/from USBD
- `[in] len` The length of the data that the application wants to transfer.

**Returns**
The length of the URBs that is available for application usage.

2.22.3.6 usbd_submit_urbs

The API to submit more than one URBs from the application.

```c
struct urb *usbd_submit_urbs(struct pipe *pp, uint16_t len)
```

Application fills the URBs and submits to USB through this API call

**Note:** This function should be protected by critical section since it is called from application's context.

**Parameters**

- `[in] pp` The pipe whose data to be transferred to/from USBD
- `[in] len` The length must be equal or smaller than the length gotten from usbd_get_app_urbs().

**Returns**
The next available URB.

2.22.3.7 usbd_pipe_process

```c
void usbd_pipe_process(struct pipe *pp)
```

The given pipe is processed to transfer the data to/from USB hardware endpoint

**Note:** This function has to be called from USBD_pipe_isr() implemented by the application.

**Parameters**

- `[in] pp` The pipe whose data to be transferred to/from USBD

2.22.3.8 usbd_pipe_purge

```c
void usbd_pipe_purge(struct pipe *pp)
```

Purge the data in the the URBs for a given pipe.

**Parameters**
2.23 DFU Device for USB Device Stack API

The file `ft900_usbd_dfu.h` contains the definitions for the USB DFU device functions in the `libft900.a` and `libft930.a` libraries.

API functions for USB Device DFU interfaces. These functions provide functionality required to communicate with a DFU application through the USB Device interface.

Please consult the Device Firmware Upgrade 1.1 Specification from the USB-IF for details of the DFU state machine employed in this driver.

2.23.1 API Cross Reference

It utilises the following library APIs:

- `ft900_gpio.h` – General Purpose I/O and Pad Control
- `ft900_sys.h` – Chip Management
- `ft900_delay.h` – Delay
- `ft900_interrupt.h` – Interrupt Management

Additional definitions are taken from:

- `ft900_usbh_internal.h` – Internal-only USB host definitions
- `ft900_usb.h` – General USB definitions
- `ft900_registers.h` – FT90x and FT93x register definitions

2.23.2 Macro Definition Documentation

2.23.2.1 `USBD_DFU_ATTRIBUTES`

```c
#define USBD_DFU_ATTRIBUTES

Value:

(USB_DFU_BMATTRIBUTES_CANDNLOAD | USB_DFU_BMATTRIBUTES_WILLDETACH | USB_DFU_BMATTRIBUTES_CANUPLOAD)
```

Sets the default feature support for the DFU library. This will allow firmware uploads (read of device firmware), downloads (program device firmware) and device detaches (no USB reset needs to be generated by the host).

2.23.2.2 `USBD_DFU_MAX_BLOCK_SIZE`

```c
#define USBD_DFU_MAX_BLOCK_SIZE

Value:

256
```

Sets the maximum size of a download or upload block for the library. The physical addresses calculated for programs are based on this value.
2.23.2.3 USBD_DFU_TIMEOUT

#define USBD_DFU_TIMEOUT

Value:
0x2000

The timeout (in milliseconds) used to revert to the appIDLE state after a DFU_DETACH request if a USB reset is not received. This is not applicable when the USB_DFU_BMATTRIBUTES_WILLDETACH bit is set in the attributes.

2.23.3 Function Documentation

2.23.3.1 USBD_DFU_timer

uint8_t USBD_DFU_timer ( void )

Decrement the detach_counter and adjusts state accordingly.

If the state is appDETACH move to dfuIDLE state if we have been in the appDETACH state longer than the attach timeout specified by the DFU_DETACH request.

**NOTE:** This is run from INTERRUPT LEVEL as a handler for an ISR.

The bmAttributes value set in the USBD_DFU_ATTRIBUTES determines the actions that are taken upon a timer event (i.e. may call a detach).

**Parameters**

attributes - The bmAttributes value set in the DFU functional descriptor. This determines the actions that are taken upon a reset.

**Returns**

Zero if timer running, non-zero if timer expired.

2.23.3.2 USBD_DFU_reset

uint8_t USBD_DFU_reset ( void )

Implementation of USB reset state handler for DFU.

Reset or advance the DFU state machine when a USB reset is encountered. This will change the state to dfuIDLE if it was in appDETACH state before. It will change to dfuERROR if a download was in progress. Otherwise it will return to appIDLE.

Return a byte to the host indicating if the next state change of the DFU state machine byte requires code to be reloaded and run. I.e. a new program needs to be run. The bmAttributes value set in the USBD_DFU_ATTRIBUTES determines the actions that are taken upon a reset.

**Returns**

status - non-zero if new program is to be run.

2.23.3.3 USBD_DFU_is_runtime

uint8_t USBD_DFU_is_runtime ( void )

Determine current mode of DFU.

**Returns**

Returns non-zero if the DFU state machine is in runtime mode.
2.23.3.4 USBD_DFU_set_dfumode

void USBD_DFU_set_dfumode ( void )

Force a transition into DFU mode. There is no detaching. This is used when the run-time mode is not used.

2.23.3.5 USBD_DFU_class_req_detach

void USBD_DFU_class_req_detach ( uint16_t timeout )

USB class request handler for DFU_DETACH.

Move the state machine to appDETACH state from appIDLE and initialise a timeout within which time the host should set a USB reset on the bus. An ACK packet is sent on the USB control IN endpoint to the host to acknowledge successful completion of this request. The bmAttributes value set in the USBD_DFU_ATTRIBUTES determines the actions that are taken upon a detach.

Parameters

[in] timeout - Number of milliseconds timeout before reverting to appIDLE if no USB reset is forthcoming from the host.

2.23.3.6 USBD_DFU_class_req_getstate

void USBD_DFU_class_req_getstate ( uint16_t requestLen )

USB class request handler for DFU_GETSTATE.

Return a single byte to the host containing the current DFU state machine byte. The data is written via the control IN endpoint to the host.

Parameters

requestLen - Number of bytes requested by the host.

2.23.3.7 USBD_DFU_class_req_getstatus

void USBD_DFU_class_req_getstatus ( uint16_t requestLen )

USB class request handler for DFU_GETSTATUS.

Return a structure to the host containing the current DFU state machine and status bytes. These are used by the application on the host to work out whether any errors have occurred and what the status of the device is. The structure is written via the control IN endpoint to the host. The bmAttributes value set in the USBD_DFU_ATTRIBUTES determines the actions that are taken upon a GET_STATUS.

Parameters

requestLen - Number of bytes requested by the host.

2.23.3.8 USBD_DFU_class_req_download

void USBD_DFU_class_req_download ( uint32_t block,
                                        uint16_t dataLength
                                        )

USB class request handler for DFU_DNLOAD.

Receive blocks of firmware from the host on the control OUT endpoint and program these into the MTP. If the state machine is in dfuIDLE then move to dfuDNLOAD_IDLE state.
If zero length data is received indicating the end of the firmware then move the state machine to dfuMANIFEST_WAIT_RESET. If an address or data length error are detected then move to the dfuERROR state. An ACK packet is sent on the USB control IN endpoint to the host to acknowledge successful completion of this request. If the bmAttributes value set in the USBD_DFU_ATTRIBUTES does not support download then this function will have no body.

Parameters

- **address** (in) - starting address of data to program. It is up to the calling program to make sure this is calculated correctly.
- **dataLength** (in) - Number of bytes to program. This can be between the control endpoint max packet size and DFU_MAX_BLOCK_SIZE.

### 2.23.3.9 USBD_DFU_class_req_upload

```c
void USBD_DFU_class_req_upload ( uint32_t block,
                                 uint16_t dataLength )
```

USB class request handler for DFU_UPLOAD.

Receive blocks of firmware from the Flash to the control IN endpoint. If the state machine is in dfuIDLE then move to dfuUPLOAD_IDLE. If an address or data length error are detected then move to the dfuERROR state. An ACK packet is sent on the USB control IN endpoint to the host to acknowledge successful completion of this request. If the bmAttributes value set in the USBD_DFU_ATTRIBUTES does not support upload then this function will have no body.

Parameters

- **address** (in) - starting address of data to read. It is up to the calling program to make sure this is calculated correctly.
- **dataLength** (in) - Number of bytes to read. This can be between the control endpoint max packet size and DFU_MAX_BLOCK_SIZE.

### 2.23.3.10 USBD_DFU_class_req_clrstatus

```c
void USBD_DFU_class_req_clrstatus ( void )
```

USB class request handler for DFU_CLRSTATUS.

Clears an error state for the DFU state machine.

### 2.23.3.11 USBD_DFU_class_req_abort

```c
void USBD_DFU_class_req_abort ( void )
```

USB class request handler for DFU_ABORT.

Aborts transaction and resets the DFU state machine.

### 2.23.3.12 USBD_DFU_is_wait_reset

```c
uint8_t USBD_DFU_is_wait_reset ( void )
```

Determine if DFU waiting to reset.

Returns
Returns non-zero if the DFU state machine is in dfuMANIFEST-WAIT-RESET and is therefore waiting for a host reset or detach/attach sequence. If the bmAttributes value set in the USBDFU_ATTRIBUTES does support manifestation then this function should not be required.

### 2.24 High Bandwidth Isochronous IN support in USB Device Stack API

The file `ft900_usbd_hbw.h` contains the definitions for the USB Device High Bandwidth Isochronous IN transfer (more than 1024 bytes and less than 3073 bytes per microframe) support APIs on FT90x Revision C onwards, in the libft900.a library.

API functions for creating a high-bandwidth isochronous IN pipe and performing high-bandwidth isochronous transfers. There is no high-bandwidth isochronous pipe support in FT93x devices.

#### 2.24.1 API Cross Reference

It utilises the following library APIs:

- `ft900_usb.h` – General USB definitions
- `ft900_registers.h` – FT90x register definitions

#### 2.24.2 Macro Definition Documentation

##### 2.24.2.1 USBD_HBW_ISOCHRONOUS_AUTOHEADER

```c
#define USBD_HBW_ISOCHRONOUS_AUTOHEADER
```

When defined, it means the UVC payload header is generated and inserted by the hardware automatically whereas the firmware only has to feed the payload data to the Isochronous IN endpoint buffer, checking the space availability in the buffer. The hardware automatically inserts the UVC header - `USB_UVC_Payload_HeaderPTS`. The PTS (presentation time stamp) engine SCR (Source clock reference) engine in the hardware can be enabled to send the Presentation time and Source clock reference in this payload header. By default the PTS engine and SCR engine are not enabled in the configuration. When end of video frame is reached, firmware has to notify the sequence end to the hardware which then automatically generates the frame end payload for UVC.

When `USBD_HBW_ISOCHRONOUS_AUTOHEADER` is disabled, the streaming firmware application has to supply the UVC header (`USB_UVC_Payload_Header` or `USB_UVC_Payload_HeaderPTS`).

#### 2.24.3 Enumeration Type Documentation

##### 2.24.3.1 USBD_HBW_HBWMODE

```c
enum USBD_HBW_HBWMODE
```

Enums used to configure whether the endpoint handles one or two or three 1024-byte packets per microframe

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD_HBW_TRANSACTION_1</td>
<td>Expect 1 ISO IN. (DATA0)</td>
</tr>
<tr>
<td>USBD_HBW_TRANSACTION_2</td>
<td>Expect 2 ISO IN. (DATA1/0)</td>
</tr>
<tr>
<td>USBD_HBW_TRANSACTION_3</td>
<td>Expect 2 ISO IN. (DATA2/1/0)</td>
</tr>
</tbody>
</table>
2.24.4 Function Documentation

2.24.4.1 USBD_HBW_init_endpoint

```c
void USBD_HBW_init_endpoint(USBD_ENDPOINT_NUMBER ep_number,
                            uint16_t fifo_size,
                            USBD_HBW_HBWMODE mode);
```

Initializes HBW pipe and hooks up to a logical endpoint. This function needs to be called after creation of IN endpoint using USBD_create_endpoint(). There is a total of 6 kB of RAM for all the endpoints EP1-7 (excluding the RAM allocated to endpoint 0). Out of 6kB, up to a maximum of 4kB of fifo size can be configured for HBW isochronous IN pipe.

**Parameters**

- `ep_number` USB IN endpoint number. (N/A for control and OUT endpoints)
- `fifo_size` Define the FIFO size for HBW ISO IN pipe allocated in SRAM
- `mode` Number of ISO IN transactions in a USB microframe (enum USBD_HBW_HBWMODE).

**Returns**

None

2.24.4.2 USBD_HBW_iso_transfer

```c
int32_t USBD_HBW_iso_transfer(USBD_ENDPOINT_NUMBER ep_number,
                               uint8_t *buffer,
                               size_t length,
                               uint8_t part,
                               size_t offset);
```

The data to be sent on the IN endpoint is copied to the FIFO in SRAM whenever at least there is 1 packet of data space is available. The offset is useful in case UVC header information is passed from the application (within the current packet) and the data following the header to be copied at an offset of header bytes.

**Parameters**

- `ep_number` USB IN endpoint number
- `buffer` Appropriately sized buffer for the transfer
- `length` the number of bytes to be sent
- `part` UNUSED
- `offset` Offset (within the current packet) from where to continue for subsequent calls when using partial packets.

**Returns**

The number of bytes actually transferred

2.24.4.3 USBD_HBW_is_fifo_full

```c
Int8_t USBD_HBW_is_fifo_full ( void )
```

Reads from HW register and indicates HBW FIFO status

**Returns**
Returns 1 if status of HBW FIFO is full. Returns 0 if not full.

2.24.4.4 USBD_HBW_is_space_avail

Int8_t USBD_HBW_is_space_avail ( void )

Reads from HW register and indicates if atleast 1 burst space (1024) available

Returns

Returns 1 if at least 1 burst space for data available. Returns 0 if not enough space.

2.24.4.5 USBD_HBW_send_end_of_frame

void USBD_HBW_send_end_of_frame ( void )

Sets SEQEND to terminate a video frame so that the hardware automatically generates the frame end payload for UVC, in case of USBD_HBW_ISOCHRONOUS_AUTOHEADER configuration.

2.25 USB Host Stack API

The file ft900_usbh.h contains the definitions for the USB host functions in the libft900.a library. This contains USB Host API function definitions, constants and structures which are exposed in the API.

2.25.1 API Cross Reference

It utilises the following library APIs:

ft900_gpio.h – General Purpose I/O and Pad Control
ft900_sys.h – Chip Management
ft900_delay.h – Delay
ft900_interrupt.h – Interrupt Management

Additional definitions are taken from:

ft900_usb.h – General USB definitions
ft900_registers.h – FT9x register definitions

2.25.2 Macro Definition Documentation

2.25.2.1 Library status values.

#define USBH_OK 0x00
Success for USB Host function.

#define USBH_ENUM_NO_CHANGE 1
No change in enumeration. This status does not constitute an error condition.

#define USBH_ENUM_PARTIAL 2
Partial enumeration only. The enumeration process did not have enough resources to completely enumerate all devices on the USB. This may constitute an error.

#define USBH_ERR_RESOURCES -1
Lack of resources to perform USB Host function.
#define USBH_ERR_USBERR -2
Host controller completed and reported an error.
#define USBH_ERR_HOST_HALTED -3
Host controller halted.
#define USBH_ERR_NOT_FOUND -4
Endpoint, Device or Interface not found.
#define USBH_ERR_REMOVED -5
Endpoint, Device or Interface removed.
#define USBH_ERR_STALLED -6
Endpoint stalled.
#define USBH_ERR_TIMEOUT -8
Endpoint transaction timeout.
#define USBH_ERR_PARAMETER -15
Request parameter error.
#define USBH_ERR_HALTED -16
Transaction completed with halted state.
#define USBH_ERR_DATA_BUF -17
Endpoint data underun or overrun.
#define USBH_ERR_BABBLE -18
Endpoint data babble detected.
#define USBH_ERR_MISSED_MICROFRAME -20
Endpoint data missed microframe.

2.25.2.2 Predefined Handles
#define USBH_ROOT_HUB_HANDLE 0
Handle used to access Root hub.
#define USBH_ROOT_HUB_PORT 0
Port used to access Root hub.
#define USBH_HUB_ALL_PORTS 0
Port used to access all devices on a hub.

2.25.3 Typedef Documentation
2.25.3.1 USBH_callback
typedef int8_t(* USBH_callback) (uint32_t id, int8_t status, uint16_t len, uint8_t *buffer)
USB callback used when completing a transaction or receiving a notification from the USBH library. It is not permissible to make a call to `USBH_transfer_async()` and specify a callback function. This will produce unspecified results.

**Parameters**

- `[in] id` Identifier for completed transaction
- `[in] status` Status of operation that caused callback
- `[in] len` Size of data buffer
- `[in] buffer` Pointer to data buffer

**Returns**

USBH_OK if the request was handled successfully.
USBH_ERR_* depending on function.

### 2.25.3.2 Device, Endpoint and Interface Handles

Handles are used to pass devices, interfaces and endpoints to the application. (Pointers to the USBH_device, USBH_interface and USBH_endpoint structures are not allowed as these are only used internally.) The handles allow embedding an enumeration value to detect stale handles which have been retained by the application. Enumeration is a dynamic operation and devices may appear and disappear without warning. This makes sure that new devices cannot be mistakenly used by an old handle.

```c
typedef uint32_t USBH_device_handle
```

Structure that is used to pass a handle to a device to the application. It is made up of a pointer to the device structure and a fairly unique value to detect enumeration changes and hence stale handles.

```c
typedef uint32_t USBH_endpoint_handle
```

Structure that is used to pass a handle to an endpoint to the application. It is made up of a pointer to the endpoint structure and a fairly unique value to detect enumeration changes and hence stale handles.

```c
typedef uint32_t USBH_interface_handle
```

Structure that is used to pass a handle to an interface to the application. It is made up of a pointer to the interface structure and a fairly unique value to detect enumeration changes and hence stale handles.

### 2.25.3.3 Endpoint Information

```c
typedef uint8_t USBH_ENDPOINT_NUMBER
```

USB Endpoint Numbers.

```c
typedef uint16_t USBH_ENDPOINT_SIZE
```

USB Endpoint Sizes.

### 2.25.4 Structure Documentation

#### 2.25.4.1 USBH_ctx

Structure containing configuration data for the USB EHCI controller, USBH memory space allocation, callback functions for USB events.

**Data Fields**

- `USBH_callback` `enumeration_change`
Field Documentation

enumeration_change

NOT CURRENTLY IMPLEMENTED. Enumeration state callback function. Optional. TBD: return a code and maybe a structure to indicate what has changed and how.

2.25.4.2 USBH_device_info

Structure containing current information about a device.

Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_number</td>
<td>uint8_t</td>
</tr>
<tr>
<td>Addr</td>
<td>uint8_t</td>
</tr>
<tr>
<td>Speed</td>
<td>uint8_t</td>
</tr>
<tr>
<td>Configuration</td>
<td>uint8_t</td>
</tr>
<tr>
<td>num_configurations</td>
<td>uint8_t</td>
</tr>
</tbody>
</table>

Field Documentation

port_number

Port number on parent hub.

addr

Configured address on USB bus.

speed

Current USB bus speed for device. Definitions in USBH_ENDPOINT_SPEED. 0 - Low speed. 1 - Full speed. 2 - High speed.

configuration

Active configuration for this device currently set with SET_CONFIGURATION.

num_configurations

Total number of configurations for this device.

2.25.4.3 USBH_interface_info

Structure containing current information about an interface.

Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_device_handle</td>
<td>dev</td>
</tr>
<tr>
<td>interface_number</td>
<td>uint8_t</td>
</tr>
<tr>
<td>alt</td>
<td>uint8_t</td>
</tr>
</tbody>
</table>

Field Documentation

dev

Handle to the parent device of this interface.

interface_number

Interface number from Interface Descriptor.

alt
Alternate setting for this interface currently set with \texttt{SET\_INTERFACE}.

\textbf{2.25.4.4 USBH\_endpoint\_info}

Structure containing current information about an endpoint.

\textbf{Data Fields}

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_interface_handle</td>
<td>iface</td>
</tr>
<tr>
<td>USBH_ENDPOINT_NUMBER</td>
<td>index</td>
</tr>
<tr>
<td>USBH_ENDPOINT_DIR</td>
<td>direction</td>
</tr>
<tr>
<td>USBH_ENDPOINT_SIZE</td>
<td>max_packet_size</td>
</tr>
<tr>
<td>USBH_ENDPOINT_TYPE</td>
<td>type</td>
</tr>
</tbody>
</table>

\textbf{Field Documentation}

\begin{itemize}
  \item \textbf{iface}
    \begin{itemize}
      \item Handle to the parent interface of this endpoint.
    \end{itemize}
  \item \textbf{index}
    \begin{itemize}
      \item Encodes USB endpoint number (0-127).
    \end{itemize}
  \item \textbf{direction}
    \begin{itemize}
      \item \textbf{IN} or \textbf{OUT} endpoint
    \end{itemize}
  \item \textbf{max\_packet\_size}
    \begin{itemize}
      \item Endpoint max packet size
    \end{itemize}
  \item \textbf{type}
    \begin{itemize}
      \item BULK, ISO, INT or CTRL endpoint
    \end{itemize}
\end{itemize}

\textbf{2.25.5 Enumeration Type Documentation}

\textbf{2.25.5.1 USBH\_STATE}

\begin{itemize}
  \item \textbf{enum USBH\_STATE}
  \begin{itemize}
    \item USB Root Hub Connection States.
  \end{itemize}
\end{itemize}

\begin{center}
\begin{tabular}{|l|l|}
  \hline
  \textbf{Enumerator} & Description \tabularnewline
  \hline
  USBH\_STATE\_NOTCONNECTED & No device is attached to USB root hub. \tabularnewline
  \hline
  USBH\_STATE\_CONNECTED & Device is attached to USB root hub. \tabularnewline
  \hline
  USBH\_STATE\_ENUMERATED & Device is attached successfully and enumerated. All downstream devices have also been successfully enumerated. \tabularnewline
  \hline
  USBH\_STATE\_ENUMERATED\_PARTIAL & Device is attached and has been partially enumerated. There may be more devices, interfaces or endpoints connected than configured. Some devices may be missing interfaces and/or endpoints. It is conceivable that some downstream devices may not be configured at \tabularnewline
  \hline
\end{tabular}
\end{center}
2.25.5.2 USBH_CONTROLLER_STATE

enum USBH_CONTROLLER_STATE

USB Host Controller State describing if the host is operational or suspending or resuming. Used to control transitions between these states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_CONTROLLER_STATE_RESET</td>
<td>Controller reset and uninitialized.</td>
</tr>
<tr>
<td>USBH_CONTROLLER_STATE_OPERATIONAL</td>
<td>Controller initialised and operational.</td>
</tr>
<tr>
<td>USBH_CONTROLLER_STATE_SUSPENDING</td>
<td>Controller performing a suspend. Transitioning from operational to suspend.</td>
</tr>
<tr>
<td>USBH_CONTROLLER_STATE_SUSPEND</td>
<td>Controller in suspend state. No SOFs generated.</td>
</tr>
<tr>
<td>USBH_CONTROLLER_STATE_RESUME</td>
<td>Controller performing a resume. Transitioning from suspend to operational.</td>
</tr>
</tbody>
</table>

2.25.5.3 USBH_ENDPOINT_DIR

enum USBH_ENDPOINT_DIR

USB Endpoint Direction.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_DIR_OUT</td>
<td>Direction host to device.</td>
</tr>
<tr>
<td>USBH_DIR_IN</td>
<td>Direction device to host.</td>
</tr>
<tr>
<td>USBH_DIR_SETUP</td>
<td>Force a SETUP PID to a control endpoint.</td>
</tr>
</tbody>
</table>

2.25.5.4 USBH_ENDPOINT_SPEED

enum USBH_ENDPOINT_SPEED

USB Endpoint Speed.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_SPEED_LOW</td>
<td>Low speed.</td>
</tr>
<tr>
<td>USBH_SPEED_FULL</td>
<td>Full speed.</td>
</tr>
<tr>
<td>USBH_SPEED_HIGH</td>
<td>High speed.</td>
</tr>
</tbody>
</table>
2.25.5.5 USBH_ENDPOINT_TYPE

enum USBH_ENDPOINT_TYPE

USB Endpoint Types.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_EP_TYPE_DISABLED</td>
<td>Disabled.</td>
</tr>
<tr>
<td>USBH_EP_BULK</td>
<td>Bulk Endpoint.</td>
</tr>
<tr>
<td>USBH_EP_INT</td>
<td>Interrupt Endpoint.</td>
</tr>
<tr>
<td>USBH_EP_ISOCH</td>
<td>Isochronous Endpoint.</td>
</tr>
<tr>
<td>USBH_EP_CTRL</td>
<td>Control Endpoint.</td>
</tr>
</tbody>
</table>

2.25.6 Function Documentation

2.25.6.1 USBH_initialise

void USBH_initialise ( USBH_ctx * ctx )

Initialise USB hardware.
Performs a software reset and initialises the USB hardware.

The USBH_ctx contains function pointers to the application for handling USB events. Currently only
and enumeration change event is implemented. This function MUST be called prior to any further
 call to the USB functions.

Parameters

[in] ctx USB context.

2.25.6.2 USBH_finalise

void USBH_finalise ( void )

Finalise USB hardware.
Releases any resources associated with the USB driver and disables the hardware.

2.25.6.3 USBHEnumerate

int8_t USBH_enumerate ( USBH_device_handle hub,
                       uint8_t port )

Force a re-enumeration of a hub.
Select a hub to force re-enumeration. To enumerate the root hub the handle is set to
USBH_ROOT_HUB_HANDLE or zero. To enumerate all ports on a hub set the port value to
USBH_HUB_ALL_PORTS or zero.

NOTE: The USBH_process call will monitor the Root hub and downstream hubs to manage the
connection/removal of devices and enumeration of devices.

Parameters
[in] hub  Handle to hub device.
[in] port Port on hub.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if hub handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.4 USBH_process

int8_t USBH_process ( void )

To be continuously called by the user application. Checks for asynchronous transfer completions and root hub events.

When a root hub connection is detected then the enumeration routine is called automatically. There is no requirement to call USBHEnumerate if USBH_process is called periodically.

Parameters
[in] Nothing

Returns
Non-zero if USB transaction has been processed.

2.25.6.5 USBH_timer

void USBH_timer ( void )

To be called every millisecond from an interrupt handler to provide timeout support for USB host transactions. This will check all pending transfers, decrement timeout values and expire any timed out transactions.

2.25.6.6 USBH_transfer

int32_t USBH_transfer ( USBH_endpoint_handle endpoint,
                        uint8_t * buffer,
                        size_t length,
                        uint16_t timeout )

Transfer data to/from a USB endpoint.
USB IN or OUT request is implied from the ep parameter. This is a blocking call to complete a transaction.

Parameters
[in] endpoint Endpoint to address.
[in] buffer Appropriately sized buffer for the transfer.
[in] length For IN transfers, the number of bytes to be sent. For OUT transfers, the maximum number of bytes to read.
[in] timeout Number of milliseconds to wait for response.
Number of bytes transferred if successful. (i.e. >= 0)
USBH_ERR_NOT_FOUND if endpoint handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.7 USBH_transfer_async

```c
int32_t USBH_transfer_async ( USBH_endpoint_handle endpoint, 
    uint8_t * buffer, 
    size_t length, 
    uint16_t timeout, 
    uint32_t id, 
    USBH_callback cb
)
```

Asynchronously transfer data to/from a USB endpoint.
USB IN or OUT request is implied from the ep parameter. This is a blocking call to complete a transaction.

**Parameters**

- **[in]** `endpoint` Endpoint to address.
- **[in]** `buffer` Appropriately sized buffer for the transfer.
- **[in]** `length` For IN transfers, the number of bytes to be sent. For OUT transfers, the maximum number of bytes to read.
- **[in]** `timeout` Number of milliseconds to wait for response. Zero for infinite timeout.
- **[in]** `id` Identifier for asynchronous transaction. Passed to the callback function.
- **[in]** `cb` Callback function to notify application of completion of asynchronous transfer. Parameters for callback function are defined in the USBH_callback typedef. The status of the transaction and any pending data (from an IN) will be returned to the callback function. The function must return with minimum processing. When it returns the USB_xfer structure is discarded and invalidated. It is not permissible to make further calls to this function from with the callback function. This will produce unspecified results. SETUP and blocking calls are allowed but may have a performance penalty on application code.

**Returns**

Number of bytes transferred if successful. (i.e. >= 0)
USBH_ERR_NOT_FOUND if endpoint handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.8 USBH_get_connect_state

```c
int8_t USBH_get_connect_state ( USBH_device_handle hub, 
    uint8_t port, 
    USBH_STATE * state
)
```
Determine if a hub port has a downstream connection.
Select a hub and a port to query. For the root hub the handle will be NULL and the port zero.

**Parameters**

- **[in]** hub Handle to hub device.
- **[in]** port Port number on hub.
- **[out]** state USBH_STATE enumeration for current state of hub port connection.

**Returns**

- USBH_OK if successful.
- USBH_ERR_NOT_FOUND if hub handle is invalid.
- USBH_ERR_* an error occurred sending the request to a USB hub.

### 2.25.6.9 USBH_get_controller_state

```c
int8_t USBH_get_controller_state ( USBH_CONTROLLER_STATE * state )
```

Get host controller state.

Get the state of the host controller. This may be used by an application to check if the controller is in suspend or operational state. There are intermediate states which can be found during transitions from operational to suspend and back. Recommended to use explicit state tests, i.e. "if state is suspended" rather than "if state is not operational".

**Parameters**

- **[out]** state State enum for host.

**Returns**

- USBH_OK if successful.

### 2.25.6.10 USBH_get_frame_number

```c
uint16_t USBH_get_frame_number ( void )
```

Get frame number.

Get the current frame number. These increments when the host is operational and will cease to increment when suspended. This number is sent in the SOF.

**Returns**

Frame number (14 bit value).

### 2.25.6.11 USBH_get_device_count

```c
int8_t USBH_get_device_count ( USBH_device_handle device,
                              uint8_t * count )
```

Get device count.

Get the count of child device enumerated for a device. For devices on the root hub the handle is set to USBH_ROOT_HUB_HANDLE.

**Parameters**

- **[in]** device Count child devices on this device
- **[out]** count Number of child devices
Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

2.25.6.12  USBH_get_device_list

int8_t USBH_get_device_list ( USBH_device_handle device,
    USBH_device_handle * child
 )

Get device list.
Get the first child device of a device. The function will return a handle to a device if there are one or more child devices. For devices on the root hub the handle is set to USBH_ROOT_HUB_HANDLE. If there are no interfaces then a NULL is returned.

Parameters
  [in]  device Handle to a device.
  [out] child Handle to first child device.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

2.25.6.13  USBH_get_next_device

int8_t USBH_get_next_device ( USBH_device_handle device,
    USBH_device_handle * next
 )

Get next device in list.
Get the next device in the list. The function will return a handle to the device if there are more devices. If there are no more devices then a NULL is returned.

Parameters
  [in]  device Handle to a device.
  [in]  device Handle to a device.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

2.25.6.14  USBH_device_get_info

int8_t USBH_device_get_info ( USBH_device_handle device,
    USBH_device_info * info
 )

Get device information.
Get information of a device.
Parameters

[in]  **device** Handle to a device.

[out] **info** Structure to receive device information.

Returns

USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

### 2.25.6.15  **USBH_device_get_descriptor**

```c
int8_t USBH_device_get_descriptor( USBH_device_handle device,
                                  uint8_t type,
                                  uint8_t index,
                                  uint16_t len,
                                  uint8_t * buf)
```

Get a descriptor from a device.
Sends a GET_DESCRIPTOR request to a device.

Parameters

[in] **device** Handle to a device.

[in] **type** Configuration descriptor type.

[in] **index** Index of descriptor.

[in] **len** Configuration descriptor len (or number of bytes to read).

[in] **buf** Location to copy descriptor into.

Returns

USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

### 2.25.6.16  **USBH_device_get_configuration**

```c
int8_t USBH_device_get_configuration( USBH_device_handle device,
                                      uint8_t * conf)
```

Gets the current configuration value of a device.
Sends a GET_CONFIGURATION request to a device.

Parameters

[in] **device** Handle to a device.
[out] conf Current configuration value.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.17 USBH_device_get_vd_pid

int8_t USBH_device_get_vd_pid (USBH_device_handle device,
                          uint16_t *vid,
                          uint16_t *pid)

Get device VID and PID.
Get the VID and PID of a device.

Parameters
[in] device Handle to a device.
[out] vid Vendor ID value from Device Descriptor.
[out] pid Product ID value from Device Descriptor.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

2.25.6.18 USBH_device_setup_transfer

int32_t USBH_device_setup_transfer (USBH_device_handle device,
                          USB_device_request *req,
                          uint8_t *buffer,
                          int16_t timeout)

Transfer data to/from a USB control endpoint.

Parameters
[in] device Device to address.
[in] req USB Device Request to send in SETUP token.
[in] buffer Appropriately sized buffer for the transfer.
[in] timeout Number of milliseconds to wait for response.

Returns
Number of bytes transferred if successful. (i.e. >= 0)
USBH_ERR_NOT_FOUND if device handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.19 USBH_device_remote_wakeup

int8_t USBH_device_remote_wakeup ( USBH_device_handle device,
        const uint8_t request
    )

Sets or clears a remote wakeup feature request to a device.
Sends a SET_FEATURE request to a device.
This function is currently NOT IMPLEMENTED.

Parameters

[in] device Handle to a device.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.20 USBH_device_set_configuration

int8_t USBH_device_set_configuration ( USBH_device_handle device,
        const uint8_t conf
    )

Sets the current configuration value of a device.
Sends a SET_CONFIGURATION request to a device.
NOTE: Should strictly only be done during enumeration.

Parameters

[in] device Handle to a device.
[in] conf New configuration value.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if the device handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.21 USBH_get_interface_count

int8_t USBH_get_interface_count ( USBH_device_handle device,
        uint8_t * count
    )
Get interface count.

Get the count of interfaces enumerated for a device.

**Parameters**

- **[in]** `device` Count interface on this device
- **[out]** `count` Number of interfaces on device

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if the device handle is invalid.

### 2.25.6.22 USBH_get_interface_list

```c
int8_t USBH_get_interface_list ( USBH_device_handle device,
                                USBH_interface_handle * interface
                          )
```

Get interface list.

Get the first interface of a device. The function will return a handle to the interface if there is one or more interfaces. If there are no interfaces then a NULL is returned.

**Parameters**

- **[in]** `device` Handle to a device.
- **[out]** `interface` Handle to the first interface.

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

### 2.25.6.23 USBH_get_next_interface

```c
int8_t USBH_get_next_interface ( USBH_interface_handle interface,
                                  USBH_interface_handle * next
                          )
```

Get next interface in list.

Get the next interface in the list. The function will return a handle to the interface if there are more interfaces. If there are no more interfaces then a NULL is returned.

**Parameters**

- **[in]** `interface` Handle to an interface.
- **[out]** `next` Handle to the next interface.

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if interface handle is invalid.

### 2.25.6.24 USBH_interface_get_info

```c
int8_t USBH_interface_get_info ( USBH_interface_handle interface,
                                  USBH_interface_info * info
                          )
```
Get interface information.
Get information of an interface.

**Parameters**

- **[in]** `interface` Handle to an interface.
- **[out]** `info` Structure to receive interface information.

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if interface handle is invalid.

### 2.25.6.25 `USBH_interface_get_class_info`

```c
int8_t USBH_interface_get_class_info ( USBH_interface_handle interface,
                                       uint8_t * devClass,
                                       uint8_t * devSubclass,
                                       uint8_t * devProtocol)
```

Get interface class, subclass and protocol.
Get the class information of an interface.

**Parameters**

- **[in]** `interface` Handle to an interface.
- **[out]** `devClass` USB class value for the interface.
- **[out]** `devSubclass` USB subclass value for the interface.
- **[out]** `devProtocol` USB protocol value for the interface.

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if interface handle is invalid.

### 2.25.6.26 `USBH_get_control_endpoint`

```c
int8_t USBH_get_control_endpoint ( USBH_device_handle device,
                                   USBH_endpoint_handle * endpoint)
```

Get control endpoint.
Get the control endpoint of a device. The function will return a handle to the control endpoint.

**Parameters**

- **[in]** `device` Handle to a device.
- **[out]** `endpoint` Handle to a control endpoint.

**Returns**
USBH_OK if successful. USBH_ERR_NOT_FOUND if device handle is invalid.

### 2.25.6.27 USBH_get_endpoint_count

```c
int8_t USBH_get_endpoint_count ( USBH_interface_handle interface,
                                 uint8_t * count )
```

Get endpoint count.
Get the count of endpoints enumerated for an interface.

**Parameters**

- **[in]** `interface` Count endpoints on this interface
- **[out]** `count` Number of endpoints on interface

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if interface handle is invalid.

### 2.25.6.28 USBH_get_endpoint_list

```c
int8_t USBH_get_endpoint_list ( USBH_interface_handle interface,
                                USBH_endpoint_handle * endpoint )
```

Get endpoint list.
Get the first endpoint of an interface. The function will return a handle to the endpoint if there are one or more endpoints. If there are no endpoints then a NULL is returned.

**Parameters**

- **[in]** `interface` Handle to an interface.
- **[out]** `next` Handle to first endpoint.

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if interface handle is invalid.

### 2.25.6.29 USBH_get_next_endpoint

```c
int8_t USBH_get_next_endpoint ( USBH_endpoint_handle endpoint,
                                USBH_endpoint_handle * next )
```

Get next endpoint in list.
Get the next endpoint in the list. The function will return a handle to the endpoint if there are more endpoints. If there are no more endpoints then a NULL is returned.

**Parameters**

- **[in]** `endpoint` Handle to an endpoint.
- **[out]** `next` Handle to next endpoint.
Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if endpoint handle is invalid.

2.25.6.30  USBH_endpoint_get_info

```c
int8_t USBH_endpoint_get_info ( USBH_endpoint_handle endpoint,
                              USBH_endpoint_info * info )
```

Get endpoint information.
Get information of an endpoint.

Parameters
- [in] **endpoint** Handle to an endpoint.
- [out] **info** Structure to receive endpoint information.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if endpoint handle is invalid.

2.25.6.31  USBH_endpoint_halt

```c
int8_t USBH_endpoint_halt ( USBH_endpoint_handle endpoint,
                            const uint8_t request )
```

Sets or clears an endpoint halt feature request to an endpoint.
Sends a SET_FEATURE request to an endpoint.

Parameters
- [in] **endpoint** Handle to an endpoint.
- [in] **request** Set or Clear Port feature. Described in Table 9-4 in Section 9.4 of USB Specification.

Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if endpoint handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.25.6.32  USBH_interface_clear_host_halt

```c
int8_t USBH_interface_clear_host_halt ( USBH_endpoint_handle endpoint )
```

Clear a halted flag on an endpoint in the host controller.
Instruct the USB host controller to remove the halt flag from an endpoint.

Parameters
- [in] **endpoint** Handle to an endpoint.
Returns
USBH_OK if successful.
USBH_ERR_NOT_FOUND if endpoint handle is invalid.

### 2.25.6.33 USBH_get_hub_status

```c
int8_t USBH_get_hub_status ( USBH_device_handle hub,
       USB_hub_status * status )
```

Return status specified hub.

For the hub pointed to by the handle, return the status of the hub. For the root hub the handle will be NULL.

Parameters

- `[in] hub` Handle to hub device.
- `[out] status` Hub status. As described in Table 11-19 and Table 11-20 of Section 11.24.2.6 in the USB Specification. Status in low word and change in high word.

Returns
USBH_OK on success.
USBH_ERR_NOT_FOUND if hub handle is invalid.
USBH_ERR_* an error occurred querying a USB hub.

### 2.25.6.34 USBH_get_hub_port_count

```c
int8_t USBH_get_hub_port_count ( USBH_device_handle hub,
       uint8_t * count )
```

Return number of ports on specified hub.

For the hub pointed to by the handle, return the number of ports that are available. For the root hub the handle will be NULL.

Parameters

- `[in] hub` Handle to hub device.
- `[out] count` Number of ports on hub.

Returns
USBH_OK on success.
USBH_ERR_NOT_FOUND if hub handle is invalid.

### 2.25.6.35 USBH_get_hub_port_status

```c
int8_t USBH_get_hub_port_status ( USBH_device_handle hub,
       const uint8_t port,
       USB_hub_port_status * status )
```

Return the status of the specified port on the hub.
For the hub pointed to by the handle, return the status of the numbered port. For the root hub the handle will be NULL.

Parameters

[in]  hub     Handle to hub device.
[in]  port    Port number on hub.
[out] status Port status. As described in Table 11-21 of Section 11.24.2.7 in the USB Specification.

Returns

USBH_OK on success.
USBH_ERR_NOT_FOUND if hub handle is invalid.
USBH_ERR_* an error occurred querying a USB hub.

2.25.6.36   USBH_hub_set/clear_feature

int8_t USBH_hub_clear_feature ( USBH_device_handle hub,
                                  const uint16_t feature
                            );

int8_t USBH_hub_set_feature ( USBH_device_handle hub,
                               const uint16_t feature
                            );

Set/Clear Features on hub.

For the hub pointed to by the handle, send a set or clear feature. For the root hub the handle will be NULL. Set or Clear feature operation described in Section 11.24.2.12 & 11.24.2.1 of the USB Specification.

Parameters

[in]  hub     Handle to hub device.
[in]  feature Port feature. As described in Table 11-17 of Section 11.24.2 in the USB Specification.

Returns

USBH_OK on success.
USBH_ERR_NOT_FOUND if hub handle is invalid.
USBH_ERR_* an error occurred sending the request to a USB hub.

2.25.6.37   USBH_hub_set/clear_port_feature

int8_t USBH_hub_set_port_feature ( USBH_device_handle hub,
                                   const uint8_t   port,
                                   const uint16_t  feature
                             );

int8_t USBH_hub_clear_port_feature ( USBH_device_handle hub,
                                     const uint8_t   port,
                                     const uint16_t  feature
                             );

Set/Clear Features on port.

For the hub pointed to by the handle, send a set or clear feature. For the root hub the handle will be NULL. Set or Clear feature operation described in Section 11.24.2.12 & 11.24.2.1 of the USB Specification.

Parameters

[in]  hub     Handle to hub device.
[in]  port    Port number on hub.
[in]  feature Port feature. As described in Table 11-17 of Section 11.24.2 in the USB Specification.

Returns

USBH_OK on success.
USBH_ERR_NOT_FOUND if hub handle is invalid.
USBH_ERR_* an error occurred sending the request to a USB hub.
Set/Clear Port Features on hub.

For the hub pointed to by the handle, send a set or clear port feature. For the root hub the handle will be NULL. Set or Clear Port feature operation described in Section 11.24.2.13 & 11.24.2.2 of the USB Specification.

Parameters

- **[in]** `hub` Handle to hub device.
- **[in]** `port` Port number on hub.
- **[in]** `feature` Port feature. As described in Table 11-17 of Section 11.24.2 in the USB Specification.

Returns

- `USBH_OK` on success.
- `USBH_ERR_NOT_FOUND` if hub handle is invalid.
- `USBH_ERR_*` an error occurred sending the request to a USB hub.
2.26 USB Host Stack Extensions API

The file `ft900_usbhx.h` contains the definitions for the USB host extension functions in the libft900.a library.

API functions for extensions to the USB Host stack. These functions provide additional functionality useful to implement a USB Host application.

2.26.1 API Cross Reference

It utilises the following library APIs:

- `ft900_usbh.h` – USB host
- Additional definitions are taken from:
  - `ft900_usb.h` – General USB definitions

2.26.2 Function Documentation

2.26.2.1 `USBHX_enumerate_wait`

```
USBH_STATE USBHX_enumerate_wait ( void )
```

Waits for a connection to the root hub and enumerates the device.

Will block until a device is connected to the root hub and then proceed to enumerate it and any downstream devices. Once this is complete then it will check the enumeration result and return. Will never return USBH_STATE_NOTCONNECTED.

**Returns**

- `USBH_STATE_CONNECTED` - a device is connected but there was a general failure to enumerate.
- `USBH_STATE_ENUMERATED` - Device connected and enumerated properly.
- `USBH_STATE_ENUMERATED_PARTIAL` - Device connected and enumeration started. Enumeration did not complete so some devices, interfaces or endpoints may be missing.

2.26.2.2 `USBHX_find_by_class`

```
int8_t USBHX_find_by_class ( USBH_device_handle * phDev,
    USBH_interface_handle * phInterface,
    uint8_t usbClass,
    uint8_t usbSubclass,
    uint8_t usbProtocol )
```

Get interface class, subclass and protocol.

Get the class information of an interface.

**Parameters**

- `[in]` **interface** Handle to an interface.
- `[out]` **class** USB class value for interface.
- `[out]` **subclass** USB subclass value for interface.
- `[out]` **protocol** USB protocol value for interface.

**Returns**
USBH_OK if successful.
USBH_ERR_NOT_FOUND if interface handle is invalid.

2.26.2.3 USBHX_find_by_vid_pid

```c
int8_t USBHX_find_by_vid_pid ( USBH_device_handle * phDev,
    uint16_t    usbVid,
    uint16_t    usbPid
)
```

Find the first device with a specific VID and PID.
Get the VID and PID of a device.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] <code>device</code></td>
<td>Handle to a device.</td>
</tr>
<tr>
<td>[out] <code>vid</code></td>
<td>Vendor ID value from Device Descriptor.</td>
</tr>
<tr>
<td>[out] <code>pid</code></td>
<td>Product ID value from Device Descriptor.</td>
</tr>
</tbody>
</table>

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.

2.26.2.4 USBHX_get_config_descriptor

```c
int8_t USBHX_get_config_descriptor ( USBH_device_handle device,
    uint8_t    type,
    uint8_t    index,
    uint16_t   offset,
    uint16_t   len,
    uint8_t *  buf
)
```

Get a partial descriptor from a device.
Sends a GET_DESCRIPTOR request to a device and returns a section of the data.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] <code>device</code></td>
<td>Handle to a device.</td>
</tr>
<tr>
<td>[in] <code>type</code></td>
<td>Configuration descriptor type.</td>
</tr>
<tr>
<td>[in] <code>index</code></td>
<td>Index of descriptor.</td>
</tr>
<tr>
<td>[in] <code>offset</code></td>
<td>Start position in descriptor to read.</td>
</tr>
<tr>
<td>[in] <code>len</code></td>
<td>Number of bytes to read from position &quot;offset&quot;.</td>
</tr>
<tr>
<td>[in] <code>buf</code></td>
<td>Location to copy descriptor into (must be minimum size of &quot;len&quot;).</td>
</tr>
</tbody>
</table>

**Returns**

USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.
USBH_OK if successful.
USBH_ERR_NOT_FOUND if device handle is invalid.
USBH_ERR_RESOURCES if there are insufficient resources.
USBH_ERR_* depending on USB bus errors.

2.26.2.5 USBHX_root_connected

int8_t USBHX_root_connected ( void )
Tests if a device is connected to the root hub.

Returns
zero - No device connected.
non-zero - A device is connected but may not be enumerated.

2.26.2.6 USBHX_root.enumerated

int8_t USBHX_root.enumerated ( void )
Tests if a device is connected to the root hub and enumerated.

Returns
zero - No device enumerated.
non-zero - A device is connected and enumerated. The device can be used with the USBH driver.

2.26.2.7 USBHX_root.enumeration.failed

int8_t USBHX_root.enumeration.failed ( void )
Tests if the enumeration worked correctly.

Returns
zero - Device(s) enumerated correctly.
non-zero - No device connected, a device is connected but not enumerated or the device may have not been enumerated completely.

2.27 HID Devices on USB Host Stack API

The file `ft900_usbhid.h` contains the definitions for the USB host HID functions in the libft900.a library.

API functions for USB Host HID devices. These functions provide functionality required to communicate with a HID device through the USB Host interface.

Please refer to the documentation produced by the USB-IF covering HID devices including the Device Class Definition for HID 1.11.

2.27.1 API Cross Reference

It utilises the following library APIs:

- `ft900_delay.h` – Delay
- `ft900_usbh.h` – USB host

Additional definitions are taken from:

- `ft900_usb.h` – General USB definitions
- `ft900_usb_hid.h` – USB HID definitions
2.27.2 Structure Documentation

2.27.2.1 USBH_HID_context

HID device context.

Holds a context structure required by each instance of the driver.

**Data Fields**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_device_handle</td>
<td>hHIDDevice</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hHIDInterface</td>
</tr>
<tr>
<td>uint8_t</td>
<td>hidInterfaceNumber</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hHIDEpIn</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hHIDEpOut</td>
</tr>
<tr>
<td>uint8_t</td>
<td>reportInSize</td>
</tr>
<tr>
<td>uint8_t</td>
<td>reportOutSize</td>
</tr>
</tbody>
</table>

**Field Documentation**

- **hHIDDevice**
  USB host device handle for HID device.

- **hHIDInterface**
  USB host interface handle for HID device.

- **hidInterfaceNumber**
  Interface number for HID device.

- **hHIDEpIn**
  Handle for IN endpoint used by HID device.

- **hHIDEpOut**
  Handle for OUT endpoint used by HID device.

- **reportInSize**
  Report size in bytes for IN reports.

- **reportOutSize**
  Report size in bytes for OUT reports.

2.27.3 Function Documentation

2.27.3.1 USBH_HID_init

```c
int8_t USBH_HID_init ( USBH_device_handle hHIDDev, 
                       USBH_interface_handle hHIDInterface, 
                       USBH_HID_context * ctx )
```

Initialise HID device.
Initialises the instance of the HID device and stores information in the USBH_HID_context structure passed from the application. This allows individual instances of the HID device to be accessed independently.

**Parameters**

- **[in]** `hHIDDev` Handle of HID device on USB bus.
- **[in]** `hHIDInterface` Handle of interface on hHIDDev to use for driver. There may be more than one HID interface on a device.
- **[out]** `ctx` Pointer to HID context.

**Returns**

Zero if successful, non-zero if not.

### 2.27.3.2 USBH_HID_get_report_size_in

```
int8_t USBH_HID_get_report_size_in ( USBH_HID_context * ctx )
```

Gets the IN report size for the HID device.

Allows the application to discover the size of reports sent by the HID device.

**Parameters**

- **[in]** `ctx` Pointer to HID context.

**Returns**

Zero if fail, non-zero for report size.

### 2.27.3.3 USBH_HID_get_report_size_out

```
int8_t USBH_HID_get_report_size_out ( USBH_HID_context * ctx )
```

Gets the OUT report size for the HID device.

Allows the application to discover the size of reports to send to the HID device.

**Parameters**

- **[in]** `ctx` Pointer to HID context.

**Returns**

Zero if fail, non-zero for report size.

### 2.27.3.4 USBH_HID_set_idle

```
int8_t USBH_HID_set_idle ( USBH_HID_context * ctx, uint16_t idle )
```

Send a SET IDLE request to the HID device.

Forms and sends a SET IDLE request to the control endpoint of the HID device. The interface number of the HID interface is included to tell the device which of possible multiple interfaces to idle.

**Parameters**

- **[in]** `ctx` Pointer to HID context.
- **[in]** `idle` Timeout for idle, zero is infinite.
Returns
Zero if successful, non-zero if not.

2.27.3.5 USBH_HID_get_report

```c
int8_t USBH_HID_get_report ( USBH_HID_context * ctx,
                                uint8_t * buffer )
```

Gets an IN report from the HID device.
Returns a report from the device's IN endpoint into the buffer pointed to in the parameters. The buffer must be large enough for the number of bytes returned in USBH_HID_get_report_size_in().

Parameters
- [in] ctx Pointer to HID context.
- [out] buffer Buffer to receive data.

Returns
Zero if successful, non-zero if not.

2.27.3.6 USBH_HID_set_report

```c
int8_t USBH_HID_set_report ( USBH_HID_context * ctx,
                                uint8_t * buffer )
```

Sends an OUT report to the HID device.
Transmits a report to the device's OUT endpoint from the buffer pointed to in the parameters. The buffer must contain at least the number of bytes returned in USBH_HID_get_report_size_out().

Parameters
- [in] ctx Pointer to HID context.
- [out] buffer Buffer providing data.

Returns
Zero if successful, non-zero if not.
2.28 BOMS Devices on USB Host Stack API

The file `ft900_usbh_boms.h` contains the definitions for the USB host BOMS functions in the `libft900.a` library.

API functions for USB Host BOMS devices. These functions provide functionality required to communicate with a BOMS device through the USB Host interface.

Please refer to the documentation produced by the USB-IF covering BOMS devices including the Mass Storage Bulk Only 1.0 specification.

2.28.1 API Cross Reference

It utilizes the following library APIs:

- `ft900_delay.h` – Delay
- `ft900_usbh.h` – USB host
- `ft900_usbhx.h` – USB host extensions

Additional definitions are taken from:

- `ft900_usb.h` – General USB definitions
- `ft900_usb_boms.h` – USB BOMS definitions

2.28.2 Macro Definition Documentation

2.28.2.1 Block Size

```c
#define USBH_BOMS_BLOCK_SIZE 512
```

Defines the size of a sector on the BOMS device used by this library. This can be 512 bytes or 2 kB according to the Bulk Only Mass Storage specification. Only 512 byte sectors have been tested.

2.28.2.2 Library Return Codes

```c
#define USBH_BOMS_OK 0
```

Success for BOMS function.

```c
#define USBH_BOMS_ERR_PARAMETER -1
```

Parameter error in call to BOMS function.

```c
#define USBH_BOMS_ERR_CLASS_NOT_SUPPORTED -2
```

Device class not supported.

```c
#define USBH_BOMS_ERR_SCSI -3
```

USB error received during SCSI commands.

```c
#define USBH_BOMS_ERR_STATUS -5
```

Error received during status phase.

```c
#define USBH_BOMS_ERR_CLASS -6
```

BOMS class error during function.

```c
#define USBH_BOMS_ERR_LUN -7
```

Requested LUN is not available.

```c
#define USBH_BOMS_ERR_CAPACITY_TIMEOUT -8
```

Requested capacity timeout.
SCSI Get Capacity request timed out.

2.28.3 Structure Documentation

2.28.3.1 **USBH_BOMS_context**

BOMS device context.

Holds a context structure required by each instance of the driver.

**Data Fields**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_device_handle</td>
<td>hBomsDevice</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hBomsInterface</td>
</tr>
<tr>
<td>uint8_t</td>
<td>bomsInterfaceNumber</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hBomsEpIn</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hBomsEpOut</td>
</tr>
<tr>
<td>uint8_t</td>
<td>maxLun</td>
</tr>
<tr>
<td>uint8_t</td>
<td>lun</td>
</tr>
<tr>
<td>uint32_t</td>
<td>lba_count</td>
</tr>
<tr>
<td>uint16_t</td>
<td>lba_size</td>
</tr>
<tr>
<td>uint16_t</td>
<td>vid</td>
</tr>
<tr>
<td>uint16_t</td>
<td>pid</td>
</tr>
<tr>
<td>uint8_t</td>
<td>vendorId [8]</td>
</tr>
<tr>
<td>uint8_t</td>
<td>productId [16]</td>
</tr>
<tr>
<td>uint8_t</td>
<td>rev [4]</td>
</tr>
<tr>
<td>uint32_t</td>
<td>tag</td>
</tr>
</tbody>
</table>

**Field Documentation**

hBomsDevice

USB host device handle for BOMS device.

hBomsInterface

USB host interface handle for BOMS device.

bomsInterfaceNumber

Interface number for BOMS device.

hBomsEpIn

Handle for IN endpoint used by BOMS device.

hBomsEpOut

Handle for OUT endpoint used by BOMS device.

maxLun

Maximum LUN supported on this BOMS device.

lun
Current LUN in use.

lba_count
Logical block count (number of sectors on device).

lba_size
Size of logical blocks (sector size).

vid
VID of BOMS device.

pid
PID of BOMS device.

vendorId
String containing Vendor Name of BOMS device (may not be NULL terminated).

productld
String containing Product Name of BOMS device (may not be NULL terminated).

rev
Device specific revision information.

tag
Library Internal Use Tag.

## 2.28.4 Function Documentation

### 2.28.4.1 USBH_BOMS_init

```c
int8_t USBH_BOMS_init ( USBH_interface_handle hBomsInterface,
                        uint8_t lun,
                        USBH_BOMS_context * ctx
                    )
```

Initialise the BOMS driver.

Setup a context for the BOMS driver to use the interfaces and settings provided in the call.

**Parameters**

- **hBomsInterface** - handle to the BOMS interface.
- **lun** - Logical Unit Number on device to use.
- **ctx** - Structure instantiated in the application to hold the context information for this instance of the driver.

**Returns**

USBH_BOMS_OK if successful

### 2.28.4.2 USBH_BOMS_get_max_lun

```c
int8_t USBH_BOMS_get_max_lun ( USBH_BOMS_context * ctx,
```

...
uint8_t * maxLun
)

Gets the number of LUNs on the BOMS device.
Queries the device to find the number of Logical Units on the device.

Parameters

  ctx - Driver context.
  maxLun - The number of the highest numbered LUN on the device.

Returns

USBH_BOMS_OK if successful

2.28.4.3 USBH_BOMS_reset

int8_t USBH_BOMS_reset ( USBH_BOMS_context * ctx )

Reset the BOMS device.
Performs a BOMS device reset operation.

Parameters

  ctx - Driver context.

Returns

USBH_BOMS_OK if successful

2.28.4.4 USBH_BOMS_read

int8_t USBH_BOMS_read ( USBH_BOMS_context * ctx,
  uint32_t lba,
  uint32_t len,
  uint8_t * buffer
)

Read sectors from the BOMS device.
Read one or more sectors from the device. This blocks until the required amount of data is read.

Parameters

  ctx - Driver context.
  lba - Logical Block Address (sector number) to commence read.
  len - Number of bytes to read. This must be a multiple of the sector size.
  buffer - Memory to receive data from on-disk sectors.

Returns

USBH_BOMS_OK if successful
USBH_BOMS_ERR_PARAMETER length is not a multiple of the sector size.
USBH_BOMS_ERR_SCSI if a SCSI (protocol error) occurred.
USBH_BOMS_ERR_STATUS the device returned a status error.
2.28.4.5 USBH_BOMS_write

```c
int8_t USBH_BOMS_write ( USBH_BOMS_context * ctx,
    uint32_t lba,
    uint32_t len,
    uint8_t * buffer
);
```

Write sectors to the BOMS device.
Write one or more sectors to the device. This blocks until the required amount of data is written.

**Parameters**
- `ctx` - Driver context.
- `lba` - Logical Block Address (sector number) to commence write.
- `len` - Number of bytes to write. This must be a multiple of the sector size.
- `buffer` - Memory to source data from.

**Returns**
- USBH_BOMS_OK if successful
- USBH_BOMS_ERR_PARAMETER length is not a multiple of the sector size.
- USBH_BOMS_ERR_SCSI if a SCSI (protocol error) occurred.
- USBH_BOMS_ERR_STATUS the device returned a status error.

2.28.4.6 USBH_BOMS_mult_read_start

```c
int8_t USBH_BOMS_mult_read_start ( USBH_BOMS_context * ctx,
    uint32_t lba,
    uint32_t len
);
```

Commence multiple sector reads from the BOMS device.
Start a read of multiple sectors from the device. The function does not block allowing data to be processed as it is read. This allows large amounts of data to be streamed from the device without using large amounts of memory to hold the data for processing.

The `USBH_BOMS_mult_read_data()` function is used to perform the read - which must take exactly the number of bytes requested - before the `USBH_BOMS_mult_end()` function completes the read. There must be no other BOMS operations while a multiple sector read operation is in process.

**Parameters**
- `ctx` - Driver context.
- `Lba` - Logical Block Address (sector number) to commence read.
- `Len` - Number of bytes to read. This must be a multiple of the sector size.

**Returns**
- USBH_BOMS_OK if successful
- USBH_BOMS_ERR_PARAMETER length is not a multiple of the sector size.
2.28.4.7 USBH_BOMS_mult_write_start

```c
int8_t USBH_BOMS_mult_write_start ( USBH_BOMS_context * ctx,
                                 uint32_t lba,
                                 uint32_t len,
                               )
```

Commence multiple sector writes to the BOMS device.

Start a write of multiple sectors to the device. The function does not block allowing data to be generated as it is written. This allows large amounts of data to be streamed to the device without using large amounts of memory to hold the data after generating it.

The USBH_BOMS_mult_write_data() function is used to perform the write – which must receive exactly the number of bytes requested – before the USBH_BOMS_mult_end() function completes the write. There must be no other BOMS operations while a multiple sector write operation is in process.

**Parameters**

- **ctx** - Driver context.
- **Lba** - Logical Block Address (sector number) to commence write.
- **Len** - Number of bytes to write. This must be a multiple of the sector size.

**Returns**

USBH_BOMS_OK if successful
USBH_BOMS_ERR_PARAMETER length is not a multiple of the sector size.

2.28.4.8 USBH_BOMS_mult_read_data

```c
int8_t USBH_BOMS_mult_read_data ( USBH_BOMS_context * ctx,
                                   uint32_t len,
                                   uint8_t * buffer,
                                 )
```

Read sectors from the BOMS device during a multiple sector read.

Reads one or more sectors from a device after a multiple sector read has been started with the USBH_BOMS_mult_read_start() function.

**Parameters**

- **ctx** - Driver context.
- **Len** - Number of bytes to read. This must be a multiple of the sector size.
- **Buffer** - Memory to receive data.

**Returns**

USBH_BOMS_OK if successful
USBH_BOMS_ERR_PARAMETER length is not a multiple of the sector size.
USBH_BOMS_ERR_SCSI if a SCSI (protocol error) occurred.
USBH_BOMS_ERR_STATUS the device returned a status error.

2.28.4.9 USBH_BOMS_mult_write_data

```c
int8_t USBH_BOMS_mult_write_data ( USBH_BOMS_context * ctx,
                                    uint32_t lba,
                                    uint32_t len
                                  )
```
uint32_t len,
uint8_t * buffer
)

Write sectors to the BOMS device during a multiple sector write.

Writes one or more sectors to a device after a multiple sector write has been started with the USBH_BOMS_mult_write_start() function.

Parameters

ctx - Driver context.

Len - Number of bytes to write. This must be a multiple of the sector size.

Buffer - Memory to source data from.

Returns

USBH_BOMS_OK if successful
USBH_BOMS_ERR_PARAMETER length is not a multiple of the sector size.
USBH_BOMS_ERR_SCSI if a SCSI (protocol error) occurred.
USBH_BOMS_ERR_STATUS the device returned a status error.

2.28.4.10 USBH_BOMS_mult_end

int8_t USBH_BOMS_mult_end ( USBH_BOMS_context * ctx )

Complete a multiple sector write or read.

Finishes a multiple sector write or read and checks the status.

Parameters

ctx - Driver context.

Returns

USBH_BOMS_OK if successful
USBH_BOMS_ERR_SCSI if a SCSI (protocol error) occurred.
USBH_BOMS_ERR_STATUS the device returned a status error.

2.28.4.11 USBH_BOMS_status

int8_t USBH_BOMS_status ( USBH_BOMS_context * ctx )

Get the device SCSI status.

Performs a SCSI Sense operation to retrieve the status of the BOMS device.

Parameters

ctx - Driver context.

Returns

USBH_BOMS_OK if successful
USBH_BOMS_ERR_SCSI if a SCSI (protocol error) occurred.
USBH_BOMS_ERR_STATUS the device returned a status error.

2.29 CDC ACM Devices on USB Host Stack API

The file ft900_usbh_cdcacm.h contains the definitions for the USB host CDC ACM functions in the libft900.a library. API functions for USB Host stack. These functions provide additional
functionality useful to implement a USB Host application. Please refer to the documentation produced by the USB-IF covering CDC devices including the Class Definitions for Communications Devices 1.2. API Cross Reference.

It utilises the following library APIs:
- `ft900_usbh.h` – USB host
- `ft900_usbxh.h` – USB host extensions

Additional definitions are taken from:
- `ft900_usb.h` – General USB definitions
- `ft900_usb_cdc.h` – USB CDC definitions

### 2.29.1 Macro Definition Documentation

#### 2.29.1.1 Feature Configuration

- `#define CDCACM_FLAG_NO_NOTIFICATION` 1
  Do not expect or poll the notification endpoint in the Communication Class Interface.
- `#define CDCACM_IN_BUFFER` 512
  Size of internal receive circular buffer for CDC DATA interface.
- `#define CDCACM_IN_MAX_PACKET` 512
  Maximum packet size of data which may be received. On High Speed devices this can be 512 bytes, on Full Speed this will be 64 bytes. The larger size will support both Full and High Speed devices.
- `#define CDCACM_NOTIFICATION_BUFFER` 12
  Size of internal buffer used to hold notifications from the CDC CONTROL interface.

#### 2.29.1.2 Library Return Codes

- `#define USBH_CDCACM_OK` 0
  Success for CDC function.
- `#define USBH_CDCACM_ERR_PARAMETER` -1
  Parameter error in call to CDC function.
- `#define USBH_CDCACM_ERR_CLASS_NOT_SUPPORTED` -2
  Device class not supported.
- `#define USBH_CDCACM_ERR_CLASS` -3
  Class request not supported.
- `#define USBH_CDCACM_ERR_DATA_ENDPOINT` -5
  Data Endpoints not found or polling failed.
- `#define USBH_CDCACM_ERR_FUNCTIONAL_DESCRIPTOR` -6
  Function descriptor not found.
- `#define USBH_CDCACM_ERR_USB` -7
  Unexpected USB error occurred.
2.29.2 Structure Documentation

2.29.2.1 USBH_CDCACM_context

CDC ACM device context.

Holds a context structure required by each instance of the driver.

### Data Fields

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBH_device_handle</td>
<td>hControlDevice</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hControlInterface</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hDataInterface</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hControlEpIn</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hDataEpIn</td>
</tr>
<tr>
<td>uint8_t</td>
<td>controlInterfaceNumber</td>
</tr>
<tr>
<td>uint8_t</td>
<td>dataInterfaceNumber</td>
</tr>
<tr>
<td>uint8_t</td>
<td>callCapabilities</td>
</tr>
<tr>
<td>uint8_t</td>
<td>acmCapabilities</td>
</tr>
<tr>
<td>USB_CDC_UartStateBitmap</td>
<td>uartState</td>
</tr>
<tr>
<td>USB_CDC_UartStateBitmap</td>
<td>networkState</td>
</tr>
<tr>
<td>int8_t</td>
<td>responseAvailable</td>
</tr>
<tr>
<td>int8_t</td>
<td>notificationStatus</td>
</tr>
<tr>
<td>uint32_t</td>
<td>notificationBuffer [CDCACM_NOTIFICATION_BUFFER/sizeof(uint32_t)]</td>
</tr>
<tr>
<td>int8_t</td>
<td>recvStatus</td>
</tr>
<tr>
<td>uint32_t</td>
<td>recvPacket [CDCACM_IN_MAX_PACKET/sizeof(uint32_t)]</td>
</tr>
<tr>
<td>uint8_t</td>
<td>recvBuffer [CDCACM_IN_BUFFER]</td>
</tr>
<tr>
<td>volatile uint16_t</td>
<td>recvBufferWrite</td>
</tr>
<tr>
<td>volatile uint16_t</td>
<td>recvBufferRead</td>
</tr>
<tr>
<td>volatile uint16_t</td>
<td>recvBufferAvail</td>
</tr>
</tbody>
</table>

### Field Documentation

- **hControlDevice**
  USB host device handle to the CDC device. There may be multiple CDC interfaces on the same devices.

- **hControlInterface**
  USB host interface handle for CDC CONTROL interface.

- **hDataInterface**
USB host interface handle for CDC DATA interface.

hControlEpIn

USB host endpoint handle for CDC CONTROL endpoint.

hDataEpIn

USB host endpoint handle for CDC DATA interface IN endpoint.

hDataEpOut

USB host endpoint handle for CDC DATA interface OUT endpoint.

controlInterfaceNumber

Interface number for CDC CONTROL interface. These are used in SETUP requests to identify the correct interface.

dataInterfaceNumber

Interface number for CDC DATA interface. These are used in SETUP requests to identify the correct interface.

callCapabilities

Call Management capabilities. Bitmap from Call Management Functional descriptor indicating the method for managing calls on the device.

acmCapabilities

Abstract Control Management capabilities. Bitmap from Abstract Control Management Functional descriptor indicating support for Comm, Line Coding, Break and Network features.

uartState

Notification bitmap for the UART state received from device.

networkState

Notification bitmap for the network state received from device.

responseAvailable

Response available notification flag indicating that an encapsulated response can be read from the interface.

notificationStatus

Last status of CDC CONTROL notification poll from USB Host driver.

notificationBuffer

Buffer to receive notification structure from USB Host driver. This must be of type uint32_t to follow alignment requirements of data buffers in USB Host driver.

recvStatus

Last status of CDC DATA IN endpoint poll from USB Host driver.

recvPacket

Buffer to receive data from the USB Host driver. This is exactly one MaxPacket size buffer. It must be of type uint32_t to follow alignment requirements of data buffers in USB Host driver.

Note: can be of type uint8_t if it is qualified with: __attribute__ ((aligned (4))).
recvBuffer
Circular buffer used to group packet data from the USB Host. This does not have any alignment issues.
recvBufferWrite
Write pointer for circular buffer (internal use).
recvBufferRead
Read pointer for circular buffer (internal use).
recvBufferAvail
Available space counters for circular buffer (internal use).

2.29.3 Function Documentation
2.29.3.1 USBH_CDCACM_init

```c
int8_t USBH_CDCACM_init ( USBH_interface_handle hControlInterface,
                           uint8_t flags,
                           USBH_CDCACM_context * ctx )
```

Initialise the CDC ACM driver.
Setup a context for the CDC driver to use the interfaces and settings provided in the call.

**Parameters**

- **hControlInterface** - handle to the CDC CONTROL interface.
- **Flags**
  - CDCACM_FLAG_NO_NOTIFICATION to not poll notification endpoint.
- **Ctx**
  - Structure instantiated in the application to hold the context information for this instance of the driver.

**Returns**

USBH_CDCACM_OK if successful

2.29.3.2 USBH_CDCACM_read

```c
int32_t USBH_CDCACM_read ( USBH_CDCACM_context * ctx,
                            uint8_t * buffer,
                            size_t len )
```

Read data from the CDC ACM device.

Read a block of data from the CDC device DATA interface. The data is buffered internally in the driver as it is produced by the CDC device and polled by the USB host. The buffer is designed to discard incoming data if the internal buffer fills. Care must therefore be taken to ensure an adequate consumption rate of data from the CDC device.
Parameters

- **ctx**: Context information for this instance of the driver.
- **Buffer**: Receiving buffer.
- **Len**: Maximum length of data to read.

Returns

Number of bytes transferred to the receiving buffer. This may be less than the amount requested if insufficient data has been received from the CDC device. A negative value will represent an error on the USB host.

### 2.29.3.3 USBH_CDCACM_write

```c
int32_t USBH_CDCACM_write ( USBH_CDCACM_context * ctx,
                            uint8_t * buffer,
                            size_t len)
```

Write data to the CDC ACM device.

Write a block of data to the CDC device DATA interface. Data is written immediately to the device without buffering.

Parameters

- **ctx**: Context information for this instance of the driver.
- **Buffer**: Transmission buffer.
- **Len**: Maximum length of data to write.

Returns

Number of bytes transferred from the transmission buffer to the device. A negative value will represent an error on the USB host.

### 2.29.3.4 USBH_CDCACM_get_poll_status

```c
void USBH_CDCACM_get_poll_status ( USBH_CDCACM_context * ctx,
                                    int8_t * notification_status,
                                    int8_t * data_status)
```

Returns the last USB Host statuses for endpoint polling.

Each time an endpoint is polled the status is stored. Both the notification endpoint and the data IN endpoint values are stored and can be queried by this function.

Parameters

- **ctx**: Context information for this instance of the driver.
- **Notification_status**: Pointer to receive status of notification endpoint polling.
- **Data_status**: Pointer to receive status of data endpoint polling.

Returns
N/A.

### 2.29.3.5 USBH_CDCACM_set_comm_feature

```c
int8_t USBH_CDCACM_set_comm_feature ( USBH_CDCACM_context * ctx,
                                          uint16_t selector,
                                          uint16_t feature)
```

Set Comm Features on the device.

The selector parameter is used to select between the abstract state and country setting data.

**Parameters**
- **ctx** - Context information for this instance of the driver.
- **Selector** - Abstract State or Country Setting: `CDC_ACM_FEATURE_SELECTOR_ABSTRACT_STATE`, `CDC_ACM_FEATURE_SELECTOR_COUNTRY_SETTING`.
- **Feature** - Bitmap to set the abstract state bitmap or country setting code.

**Returns**
- `USBH_CDCACM_OK` – if the interface supports the COMM Feature requests.
- `USBH_CDCACM_ERR_CLASS` – if it does not support it.

### 2.29.3.6 USBH_CDCACM_clear_comm_feature

```c
int8_t USBH_CDCACM_clear_comm_feature ( USBH_CDCACM_context * ctx,
                                          uint16_t selector)
```

Clear the Comm Features on the device.

The selector parameter is used to select between the abstract state and country setting data.

**Parameters**
- **ctx** - Context information for this instance of the driver.
- **Selector** - Abstract State or Country Setting: `CDC_ACM_FEATURE_SELECTOR_ABSTRACT_STATE`, `CDC_ACM_FEATURE_SELECTOR_COUNTRY_SETTING`.

**Returns**
- `USBH_CDCACM_OK` – if the interface supports the COMM Feature requests.
- `USBH_CDCACM_ERR_CLASS` – if it does not support it.

### 2.29.3.7 USBH_CDCACM_get_acm_capabilities

```c
int8_t USBH_CDCACM_get_acm_capabilities ( USBH_CDCACM_context * ctx )
```

Returns the ACM Capabilities bitmap.

Returns the bmCapabilities value obtained from the Abstract Control Management Functional Descriptor for the CDC ACM interface.

**Parameters**
ctx - Context information for this instance of the driver.

Returns
Values are defined as CDC_ACM_CAPABILITIES_*

2.29.3.8 USBH_CDCACM_get_call_capabilities

int8_t USBH_CDCACM_get_call_capabilities ( USBH_CDCACM_context * ctx )

Returns the Call Capabilities bitmap.

Returns the bmCapabilities value obtained from the Call Management Functional Descriptor for the CDC ACM interface.

Parameters
ctx - Context information for this instance of the driver.

Returns
Values are defined as CDC_CM_CAPABILITIES_*

2.29.3.9 USBH_CDCACM_get_comm_feature

int8_t USBH_CDCACM_get_comm_feature ( USBH_CDCACM_context * ctx, uint16_t selector, uint16_t * feature )

Get a bitmap containing the currently set Comm Features.
The selector parameter is used to select between the abstract state and country setting data.

Parameters
ctx - Context information for this instance of the driver.
Selector - Abstract State or Country Setting:
CDC_ACM_FEATURE_SELECTOR_ABSTRACT_STATE,
CDC_ACM_FEATURE_SELECTOR_COUNTRY_SETTING.

Feature - Bitmap to receive abstract state bitmap or country setting code.

Returns
USBH_CDCACM_OK – if the interface supports the COMM Feature requests.
USBH_CDCACM_ERR_CLASS – if it does not support it.

2.29.3.10 USBH_CDCACM_get_encapsulated_response

int8_t USBH_CDCACM_get_encapsulated_response ( USBH_CDCACM_context * ctx, char * rsp, uint16_t len )

Read an encapsulated command response from the CDC ACM device.

Read a block of data from the CDC device control interface. Data is read immediately to the device without buffering.
Parameters

ctx - Context information for this instance of the driver.
Rsp - receive buffer for receiving response.
Len - Maximum length of data to receive.

Returns
Number of bytes transferred from the device to the receive buffer.

2.29.3.11  USBH_CDCACM_get_line_coding

int8_t USBH_CDCACM_get_line_coding ( USBH_CDCACM_context * ctx,
                                      USB_CDC_line_coding * coding )

Get Current Line Coding settings from the device.
The USB_CDC_line_coding structure describes the data output format on the 'UART' side of the
CDC device. This will query the settings and return them in the structure.

Parameters

ctx - Context information for this instance of the driver.
Coding - Pointer to structure containing the currently set parameters for formatting data.

Returns
USBH_CDCACM_OK – if the interface supports the Line Coding Feature requests.
USBH_CDCACM_ERR_CLASS – if it does not support it.

2.29.3.12  USBH_CDCACM_set_line_coding

int8_t USBH_CDCACM_set_line_coding ( USBH_CDCACM_context * ctx,
                                      USB_CDC_line_coding * coding )

Set Line Coding on the device.
The USB_CDC_line_coding structure is used to set the data output format on the CDC device.
This is on the ‘UART’ side of the device.

Parameters

ctx - Context information for this instance of the driver.
Coding - Pointer to structure containing the requested parameters for formatting data.

Returns
USBH_CDCACM_OK – if the interface supports the Line Coding Feature requests.
USBH_CDCACM_ERR_CLASS – if it does not support it.

2.29.3.13  USBH_CDCACM_get_network_connection

uint8_t USBH_CDCACM_get_network_connection ( USBH_CDCACM_context * ctx,
                                             USB_CDC_NetworkConnectionBitmap * state )
Returns the Network state bitmap. When a notification arrives updating the Network state this is kept in the driver and can be queried by this command.

**Parameters**

- **ctx** - Context information for this instance of the driver.
- **State** - Pointer to structure to receive last received state.

**Returns**

USBH_CDCACM_OK – if the interface supports the Network_Connection notification.
USBH_CDCACM_ERR_CLASS – if it does not support it.

### 2.29.3.14 **USBH_CDCACM_get_uart_state**

```c
uint8_t USBH_CDCACM_get_uart_state (USBH_CDCACM_context *ctx,
                                      USB_CDC_UartStateBitmap *state)
```

Returns the UART state bitmap. When a notification arrives updating the UART state this is kept in the driver and can be queried by this command.

**Parameters**

- **ctx** - Context information for this instance of the driver.
- **State** - Pointer to structure to receive last received state.

**Returns**

USBH_CDCACM_OK – if the interface supports the Serial_State notification.
USBH_CDCACM_ERR_CLASS – if it does not support it.

### 2.29.3.15 **USBH_CDCACM_set_control_line_state**

```c
int8_t USBH_CDCACM_set_control_line_state (USBH_CDCACM_context *ctx,
                                          USB_CDC_control_line_state *state)
```

Set Control Line State on the device.
The **USB_CDC_control_line_state** structure is used to set the state of the control lines on the CDC device. This is on the “UART” side of the device.

**Parameters**

- **ctx** - Context information for this instance of the driver.
- **State** - Pointer to structure containing the requested state for the control lines.

**Returns**

USBH_CDCACM_OK – if the interface supports the Line Coding Feature requests.
USBH_CDCACM_ERR_CLASS – if it does not support it.
2.29.3.16  USBH_CDCACM_get_response_available

int8_t USBH_CDCACM_get_response_available ( USBH_CDCACM_context * ctx )

Indicates the response to an encapsulated command is waiting.
When a notification arrives indicating that a response to a encapsulated command is waiting to be
read this is kept in the driver and can be queried by this command.

Parameters

cctx - Context information for this instance of the driver.

Returns
Non-zero if a encapsulated response is available.

2.29.3.17  USBH_CDCACM_send_encapsulated_command

int8_t USBH_CDCACM_send_encapsulated_command ( USBH_CDCACM_context * ctx,
                                               char * cmd,
                                               uint16_t len
                                      )

Send an encapsulated command to the CDC ACM device.
Write a block of data to the CDC device control interface. Data is written immediately to the device
without buffering.

Parameters

cctx - Context information for this instance of the driver.

Cmd  - Transmission buffer containing command.

Len  - Maximum length of data to write.

Returns
Number of bytes transferred from the transmission buffer to the device.

2.29.3.18  USBH_CDCACM_send_break

int8_t USBH_CDCACM_send_break ( USBH_CDCACM_context * ctx,
                                uint16_t duration
                        )

Instructs the device to set a break state on the UART line.

Parameters

cctx  - Context information for this instance of the driver.

Duration  - Length of time in milliseconds to set the break state.

Returns
USBH_CDCACM_OK – if the interface supports the Send_Break request.
USBH_CDCACM_ERR_CLASS – if it does not support it.
2.30 Android Open Accessory (AOA) Devices on USB Host Stack API

The file `ft900_usbh_aoa.h` contains the definitions for the USB host AOA functions in the `libft900.a` library.

Please refer to the Android AOA Documentation at: https://source.android.com/devices/accessories/aoa2.html for more details on AOA features and protocol.

2.30.1 API Cross Reference

It utilizes the following libraries:

- `ft900_usbh.h` – USB host
- `ft900_usbhx.h` – USB host extensions

Additional definitions are taken from:

- `ft900_usb.h` – General USB definitions
- `ft900_usb_aoa.h` – AOA definitions

2.30.2 Macro Definition Documentation

2.30.2.1 Library Return Codes

```c
#define USBH_AOA_OK 0
Success for AOA function.
#define USBH_AOA_DETECTED 1
AOA device detected in accessory mode. Device is ready to use.
#define USBH_AOA_STARTED 2
AOA device detected in accessory mode. Device has been restarted and need re-enumerated.
#define USBH_AOA_ERR_PARAMETER -1
Parameter error in call to AOA function.
#define USBH_AOA_ERR_CLASS_NOT_SUPPORTED -2
Device class not supported.
#define USBH_AOA_ERR_CLASS -3
Class request not supported.
#define USBH_AOA_ERR_PROTOCOL -4
AOA Protocol not supported.
#define USBH_AOA_ERR_CONFIG -5
AOA Protocol configuration error.
#define USBH_AOA_ERR_ENDPOINT -6
Data Endpoints not found or polling failed.
#define USBH_AOA_ERR_USB -7
```

Unexpected USB error occurred.

2.30.3 Structure Documentation

2.30.3.1 USBH_AOA_context

AOA device context.

Holds a context structure required by each instance of the driver.

**Data Fields**

<table>
<thead>
<tr>
<th>USBH_device_handle</th>
<th>hDevAccessory</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t</td>
<td>vid</td>
</tr>
<tr>
<td>uint16_t</td>
<td>pid</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hAccessoryInterface</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hAccessoryEpIn</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hAccessoryEpOut</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hAdbInterface</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hAdbEpIn</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hAdbEpOut</td>
</tr>
<tr>
<td>USBH_interface_handle</td>
<td>hAudioInterface</td>
</tr>
<tr>
<td>USBH_endpoint_handle</td>
<td>hAudioEp</td>
</tr>
<tr>
<td>uint16_t</td>
<td>maxIsoSize</td>
</tr>
<tr>
<td>uint16_t</td>
<td>protocol</td>
</tr>
<tr>
<td>uint16_t</td>
<td>HIDDescriptorSize</td>
</tr>
</tbody>
</table>

**Field Documentation**

- hDevAccessory
  Handle to the AOA device.
- Vid, pid
  VID and PID of device when in accessory mode
- hAccessoryInterface
  Interface handle for AOA general interface.
- hAccessoryEpIn
  Endpoint handles for the AOA general interface.
- hAdbInterface
  Interface handle for AOA adb interface
- hAdbEpIn, hAdbEpOut
  Endpoint handles for the AOA adb interface.
- hAudioInterface
  Interface handle for AOA audio interface.
hAudioEp
Audio device endpoint handle.
maxIsoSize
Maximum packet size of audio isochronous endpoint.
Protocol
Android Open Accessory Protocol version.
HIDDescriptorSize
Size of HID descriptor.

2.30.4 Function Documentation

2.30.4.1 USBH_AOA_init

int8_t USBH_AOA_init ( USBH_interface_handle hAOAInterface,
                       USBH_AOA_context *ctx,
                       USBH_AOA_descriptors * descriptors,
                       int16_t audio
                     )

Initialise the AOA driver.
The sequence for connecting to AOA devices is:
1) Check AOA protocol is valid. This means that the special vendor SETUP command works and the
   return value is non-zero and matches a protocol version supported by the driver.
2) Send string descriptors to the AOA device using the vendor SETUP commands. Then send a
   start accessory device vendor SETUP command.
3) The device re-enumerates by doing a device reset.
4) The host re-enumerates the device as an Android accessory.
The device will need to be attached using USBH_AOA_attach() once it has been re-enumerated.

Parameters

- **hAOAInterface** - handle to the AOA interface to use.
- **Descriptors** - Pointer to a structure containing string descriptors to send to the AOA device.
- **Ctx** - Structure instantiated in the application to hold the context information for this instance of the driver.
- **Audio** - if the protocol supports audio then send the enable audio command to the
  accessory to enable the audio interface

Returns

USBH_AOA_STARTED if an Android accessory device in its normal mode was detected. It will have
been started as an accessory and will therefore perform a device reset and be re-enumerated.
USBH_AOA_DETECTED if an Android accessory device in accessory mode was detected. This
device can now be attached and used. USBH_AOA_ERR_CLASS_NOT_SUPPORTED if a device which is not an Android accessory was detected.

2.30.4.2 USBH_AOA_attach

int8_t USBH_AOA_attach ( USBH_AOA_context *ctx)

Attaches to the AOA device.

Connects to the AOA device which is in accessory mode. It will decode the AOA protocol, VID and the PID to determine support for accessories, adb bridge and audio. Endpoints and size information is stored for use by the driver later.

Parameters

ctx - context of AOA device to use.

Returns

USBH_AOA_OK if successful
USBH_AOA_ERR_CONFIG if a device reporting to possess a particular interface does not, in fact, present that interface. USBH_AOA_ERR_CLASS_NOT_SUPPORTED if a device which is not an Android accessory was detected.

2.30.4.3 USBH_AOA_detach

int8_t USBH_AOA_detach ( USBH_AOA_context *ctx)

Detaches from the AOA device.

Parameters

ctx - context of AOA device to use.

Returns

USBH_AOA_OK if successful

2.30.4.4 USBH_AOA_get_protocol

int8_t USBH_AOA_detach ( USBH_AOA_context *ctx)

Detaches from the AOA device.

Parameters

ctx - Context of AOA device to use.

Protocol - pointer to BCD protocol revision

Returns

This will return a positive value if the device supports the accessory class. Zero if it does not. Negative if there was an error.

2.30.4.5 USBH_AOA_has_accessory

int8_t USBH_AOA_has_accessory ( USBH_AOA_context *ctx)

AOA device supports accessories.

Parameters

ctx - Context of AOA device to use.

Returns
This will return a positive value if the device supports the accessory class. Zero if it does not. Negative if there was an error.

### 2.30.4.6 USBH_AOA_has_audio

```c
int8_t USBH_AOA_has_audio ( USBH_AOA_context *ctx)
```

AOA device supports audio.

**Parameters**

- **ctx** - Context of AOA device to use.

**Returns**

This will return a positive value if the device supports the audio class. Zero if it does not. Negative if there was an error.

### 2.30.4.7 USBH_AOA_has_adb

```c
int8_t USBH_AOA_has_adb ( USBH_AOA_context *ctx)
```

AOA device supports adb.

**Parameters**

- **ctx** - Context of AOA device to use.

**Returns**

This will return a positive value if the device supports the adb class. Zero if it does not. Negative if there was an error.

### 2.30.4.8 USBH_AOA_get_audio_endpoint

```c
int8_t USBH_AOA_get_audio_endpoint ( USBH_AOA_context *ctx,
                                         USBH_endpoint_handle * hAudio,
                                         uint16_t * maxSize)
```

Gets a handle to the audio endpoint. If the AOA device supports it then the audio isochronous endpoint can be obtained with this call.

**Parameters**

- **ctx** - Context of AOA device to use.
- **hAudio** - Pointer to handle to receive audio endpoint.
- **maxSize** - Max size of the audio endpoint

**Returns**

USBH_AOA_OK if successful.

### 2.30.4.9 USBH_AOA_register_hid

```c
int8_t USBH_AOA_register_hid ( USBH_AOA_context *ctx,
                                uint16_t hidID,
                                uint16_t descriptorSize)
```


Register a new HID device.

**Parameters**

ctx  - Context of AOA device to use.

hidID  - ID of new HID device

descriptorSize  - Number of bytes in HIDs report descriptor. This is sent separately as it can be changed.

**Returns**

USBH_AOA_OK if successful.

### 2.30.4.10  USBH_AOA_unregister_hid

```c
int8_t USBH_AOA_unregister_hid ( USBH_AOA_context *ctx, uint16_t hidID )
```

Unregister a HID device.

**Parameters**

ctx  - Context of AOA device to use.

hidID  - ID of previously registered HID device

**Returns**

USBH_AOA_OK if successful.

### 2.30.4.11  USBH_AOA_set_hid_report_descriptor

```c
int8_t USBH_AOA_set_hid_report_descriptor ( USBH_AOA_context *ctx, uint16_t hidID, uint16_t descriptorOffset, uint16_t descriptorLength, uint8_t *descriptor )
```

Set the HID descriptor for a HID device.

**Parameters**

ctx  - Context of AOA device to use.

hidID  - ID of previously registered HID device

descriptorOffset  - used when the HID descriptor is sent in multiple packets. This is the offset to the position in the descriptor where this fragment goes.

descriptorLength  - Length of this section of HID descriptor. If the HID descriptor is sent in one packet then this will be the same as the length set in descriptorSize parameter of USBH_AOA_register_hid. If the descriptor
is made up of multiple packets then the length will be smaller.

Descriptor
- HID descriptor.

2.30.4.12   USBH_AOA_send_hid_data

```c
int8_t USBH_AOA_send_hid_data ( USBH_AOA_context *ctx,
                                   uint16_t     hidID,
                                   uint16_t     reportSize,
                                   uint8_t *    data,
                               )
```

Send a report descriptor to the AOA device.

Parameters
- `ctx` - Context of AOA device to use.
- `hidID` - ID of previously registered HID device
- `reportSize` - Number of bytes in HIDs report.
- `data` - Report data.

2.31 FT devices on USB host stack API (ft900_usbh_ft.h)

The file ft900_usbh_ft.h contain the API functions for enumerating FT devices on USB Host stack. These functions provide additional functionality useful to implement a USB Host application.

2.31.1 API Cross Reference

It utilizes the following libraries:
- `ft900_usbh.h` – USB host

2.31.2 Structure Documentation

2.31.2.1 USBH_FT232_context

Holds a context structure required by each instance of the FT232 driver.

- `USBH_device_handle hDevice` - Handle to the FT232 device. There may be multiple FT232 interfaces on the same devices.
- `USBH_interface_handle hDataInterface` - Interface handles for FT232 DATA interface
- `USBH_endpoint_handle hDataEpIn` - IN Endpoint handle for the FT232 DATA interfaces.
- `USBH_endpoint_handle hDataEpOut` - OUT Endpoint handle for the FT232 DATA interfaces.
- `uint16_t bcdDev` - bcdDevice from the Device Descriptor. Used to work out the type of FT232 device we are connected to.
- `uint8_t dataInterfaceNumber` - Interface number for data interface. These are used in SETUP requests to identify the correct interface.
- `int8_t recvStatus` - Last status of FT232 DATA IN endpoint poll from USB Host driver
uint32_t recvPacket

Buffer to receive data from USB Host driver. This is exactly one MaxPacket size buffer. It must be of type uint32_t to follow alignment requirements of data buffers in USB Host driver. Note: can be of type uint8_t if it is qualified with: attribute ((aligned (4)))

uint8_t recvBuffer

Circular buffer used to group packet data from USB Host. This does not have any alignment issues.

uint16_t recvBufferWrite

Read pointers for circular buffer.

uint16_t recvBufferRead

Write pointers for circular buffer.

uint16_t recvBufferAvail

Avail pointers for circular buffer.

uint16_t lastModemStatus

Modem status and line status from the device. The least significant byte of the modemstat parameter holds the modem status. The line status is in the most significant byte.

2.31.3 Functions

2.31.3.1 USBH_FT232_init

int8_t USBH_FT232_init (USBH_interface_handle hInterface, uint8_t flags, USBH_FT232_context *ctx)

Initialises the FT232 driver. Setup a context for the FT232 driver to use the interface and settings provided in the call.

Parameters:

USBH_interface_handle hInterface - handle to the FT232 interface to use.

uint8_t flags - None.

USBH_FT232_context * ctx - structure instantiated in the application to hold the context information for this instance of the driver.

Returns:

USBH_FT232_OK if successful

2.31.3.2 USBH_FT232_read

int32_t USBH_FT232_read (USBH_FT232_context *ctx, uint8_t *buffer, size_t len)

Reads a block of data from the FT232 device DATA interface. The data is buffered internally in the driver as it is produced by the FT232 device and polled by the USB host. The buffer is designed to discard incoming data if the internal buffer fills. Care must therefore be taken to ensure an adequate consumption rate of data from the FT232 device.

Parameters:

USBH_FT232_context * ctx - context information for this instance of the driver.

uint8_t * buffer - receiving buffer.

size_t len - maximum length of data to read.

Returns:

Number of bytes transferred to the receiving buffer. This may be less than the amount requested if insufficient data has been received from the CDC device.
2.31.3.3 USBH_FT232_write

```c
int32_t USBH_FT232_write (USBH_FT232_context *ctx, uint8_t *buffer, size_t len)
```

Writes a block of data to the CDC device DATA interface. Data is written immediately to the device without buffering.

**Parameters:**
- `USBH_FT232_context * ctx`: context information for this instance of the driver.
- `uint8_t * buffer`: transmission buffer.
- `size_t len`: maximum length of data to write.

**Returns:**

Number of bytes transferred from the transmission buffer to the device.

2.31.3.4 USBH_FT232_set_baud_rate

```c
int8_t USBH_FT232_set_baud_rate (USBH_FT232_context *ctx, uint32_t baud)
```

Sets FT232 Baud Rate. The baud rate is passed as a `uint32_t` and the routine works out the divisor and sub-integer prescaler required. Refer to: [http://www.ftdichip.com/Support/Documents/AppNotes/AN232B-05_BaudRates.pdf](http://www.ftdichip.com/Support/Documents/AppNotes/AN232B-05_BaudRates.pdf) It doesn't check if the baud rate can be calculated within the +/- 3% required to ensure a stable link.

**Parameters:**
- `USBH_FT232_context * ctx`: context information for this instance of the driver.
- `uint32_t baud`: requested baud rate.

**Returns:**

- `USBH_FT232_OK` - if the interface supports the COMM Feature requests.
- `USBH_FT232_ERR_CLASS` - if it does not support it.

2.31.3.5 USBH_FT232_set_flow_control

```c
int8_t USBH_FT232_set_flow_control (USBH_FT232_context *ctx, uint16_t mode)
```

Sets FT232 Flow Control. Flow control can be set as CTS/RTS, DTR/DSR or None.

**Parameters:**
- `USBH_FT232_context * ctx`: context information for this instance of the driver.
- `uint16_t mode`: flow control mode required. Can be zero for none or `USB_FT232_SETFLOWCTRL_RTS_CTS` or `USB_FT232_SETFLOWCTRL_DTR_DSR`.

**Returns:**

- `USBH_FT232_OK` - if the interface supports the COMM Feature requests.
- `USBH_FT232_ERR_CLASS` - if it does not support it.

2.31.3.6 USBH_FT232_set_modem_control

```c
int8_t USBH_FT232_set_modem_control (USBH_FT232_context *ctx, uint16_t mode, uint8_t assert)
```
Sets FT232 Modem Control. Enable RTS, DTR signals for use with flow control and set their current state.

**Parameters:**

- \texttt{USBH_FT232_context * ctx} - context information for this instance of the driver.
- \texttt{uint16_t mode} - flow control mode \texttt{USB_FT232_SETFLOWCTRL_RTS_CTS} or \texttt{USB_FT232_SETFLOWCTRL_DTR_DSR}
- \texttt{uint8_t assert} - To set or clear RTS or DTR control signals according to the flow control selected. Value 1 to Set RTS or DTR and Value 0 to clear RTS or DTR.

**Returns:**

- \texttt{USBH_FT232_OK} - if the interface supports the COMM Feature requests.
- \texttt{USBH_FT232_ERR_CLASS} - if it does not support it.

**2.31.3.7 USBH_FT232_set_data**

\texttt{int8_t USBH_FT232_set_data (USBH_FT232_context *ctx, uint16_t bits, uint16_t parity, uint16_t stop)}

Sets FT232 Data Format. Data format sets the number of data bits, stop bits and parity mode used.

**Parameters:**

- \texttt{USBH_FT232_context * ctx} - context information for this instance of the driver.
- \texttt{uint16_t bits} - Can be either \texttt{USB_FT232_SETDATA_7_BIT} or \texttt{USB_FT232_SETDATA_8_BIT}
- \texttt{uint16_t parity} - Can be one of \texttt{USB_FT232_SETDATA_NOPAR}, \texttt{USB_FT232_SETDATA_ODDPAR}, \texttt{USB_FT232_SETDATA_EVENPAR}, \texttt{USB_FT232_SETDATA_MARKPAR}, \texttt{USB_FT232_SETDATA_SPACEPAR}.
- \texttt{uint16_t stop} - Number of stop bits. Can be one of \texttt{USB_FT232_SETDATA_1_STOP} or \texttt{USB_FT232_SETDATA_2_STOP}.

**Returns:**

- \texttt{USBH_FT232_OK} - if the interface supports the COMM Feature requests.
- \texttt{USBH_FT232_ERR_CLASS} - if it does not support it.

**2.31.3.8 USBH_FT232_set_latency**

\texttt{int8_t USBH_FT232_set_latency (USBH_FT232_context *ctx, uint16_t latency)}

Sets FT232 Latency Timer. Latency timer can be set from 2 upwards.

**Parameters:**

- \texttt{USBH_FT232_context * ctx} - context information for this instance of the driver.
- \texttt{uint16_t latency} - Number of frames between reporting by FT232 device.

**Returns:**

- \texttt{USBH_FT232_OK} - if the interface supports the COMM Feature requests.
- \texttt{USBH_FT232_ERR_CLASS} - if it does not support it.
USBH_FT232_OK - if the interface supports the COMM Feature requests. USBH_FT232_ERR_CLASS - if it does not support it.

### 2.3.1.3.9 `USBH_FT232_get_poll_status`

```c
void USBH_FT232_get_poll_status (USBH_FT232_context *ctx, int8_t *data_status)
```

Returns the last USB Host statuses for endpoint polling. Each time an endpoint is polled the status is stored. The data IN endpoint values are stored and can be queried by this function.

**Parameters:**
- `USBH_FT232_context * ctx` - context information for this instance of the driver.
- `int8_t * data_status` - pointer to receive status of data endpoint polling.

**Returns:**
None

### 2.3.1.10 `USBH_FT232_get_latency`

```c
int8_t USBH_FT232_get_latency (USBH_FT232_context *ctx, uint16_t *latency)
```

Gets FT232 Latency Timer. Latency timer can be got from 2ms upwards.

**Parameters:**
- `USBH_FT232_context * ctx` - context information for this instance of the driver.
- `uint16_t * latency` - Number of frames between reporting by FT232 device.

**Returns:**
- USBH_FT232_OK - if the interface supports the COMM Feature requests.
- USBH_FT232_ERR_CLASS - if it does not support it.

### 2.3.1.11 `USBH_FT232_get_modemstat`

```c
int8_t USBH_FT232_get_modemstat (USBH_FT232_context *ctx, uint16_t *modemstat)
```

Gets the modem status and line status from the device. The least significant byte of the modemstat parameter holds the modem status. The line status in most significant byte. The modem status is bit-mapped as follows: Clear To Send (CTS) = 0x10, Data Set Ready (DSR) = 0x20, Ring Indicator (RI) = 0x40, Data Carrier Detect (DCD) = 0x80. The line status is bit-mapped as follows: Overrun Error (OE) = 0x02, Parity Error (PE) = 0x04, Framing Error (FE) = 0x08, Break Interrupt (BI) = 0x10.

**Parameters:**
- `USBH_FT232_context * ctx` - context information for this instance of the driver.
- `uint16_t * modemstat` - Pointer to a variable which receives the modem status and line status from the device.

**Returns:**
- USBH_FT232_OK - if the interface supports the COMM Feature requests.
- USBH_FT232_ERR_CLASS - if it does not support it.
2.31.3.12 USBH_FT232_eeprom_read

```c
int8_t USBH_FT232_eeprom_read (USBH_FT232_context *ctx, uint16_t e2address, uint16_t *e2data)
```

Read a value from an EEPROM location. EEPROMs for FTDI devices are organized by WORD, so each value returned is 16-bits wide.

**Parameters:**

- `USBH_FT232_context * ctx` - context information for this instance of the driver.
- `uint16_t e2address` - EEPROM location to read from.
- `uint16_t * e2data` - Pointer to the WORD value read from the EEPROM.

**Returns:**

- `USBH_FT232_OK` - if the interface supports the COMM Feature requests.
- `USBH_FT232_ERR_CLASS` - if it does not support it.

### 2.32 Startup DFU Feature

The file `ft900_startup_dfu.h` contains the definitions for the USB start up DFU device functions in the libft900.a and libft930.a libraries.

The Startup DFU library allows an application to enable the USB device on the FT9xx temporarily to present a DFU interface to the USB host. Software on the USB host can then update the application stored in Flash on the FT9xx regardless of the functionality or features of the existing application.

The feature can be added to any application by adding a call to the function `STARTUP_DFU`. This call can be made under any conditions – maybe a button press at power-up detected by a GPIO or just unconditionally when the application is started.

The USB interface remains active for a short period of time (~200ms) and once activated by enumeration from the USB host will continue to stay active for around 1000ms after activity has ceased.

This file contains Startup DFU feature function definitions, constants and structures which are exposed in the API.

Note that as this is a USB device all transaction nomenclature is from the point of view from the host. If the device sends data to the host then it is called an IN transaction, if it receives data from the host then it is an OUT transaction.

### 2.32.1 API Cross Reference

It utilises the following library APIs:

- `ft900_timers.h` – Timers
- `ft900_sys.h` – Chip Management
- `ft900_interrupt.h` – Interrupt Management
- `ft900_usb.h` – USB device
- `ft900_usb_dfu.h` – USB DFU definitions

Additional definitions are taken from:

- `ft900_usb.h` – General USB definitions
- `ft900_usb_dfu.h` – USB DFU definitions
2.32.2 Macro Definition Documentation

2.32.2.1 STARTUP_DFU

#define STARTUP_DFU(...) 
Macros to overload startup_dfu function. Allows the STARTUP_DFU call to be made with either no parameters or with one parameter. This permits an optional timeout to be passed to the startup_dfu() function.

2.32.3 Function Documentation

2.32.3.1 STARTUP_DFU

void startup_dfu ( int timeout )
Temporarily start the USB device with a DFU interface.
When called, the USB device will be enabled for around 200ms allowing a USB host to enumerate the device. Once enumerated, the function will wait for around 1000ms for a DFU connection from the USB host. This will allow a program on the host PC to download new firmware to the device. The function returns after one of the timeouts has completed. If the firmware on the device is updated or the device is reset via a USB reset then the device will be reset.

Parameters

int timeout 
Number of milliseconds to wait until a connection from a host controller is established and a DFU_DETACH request sent to the device. A value of zero will result in the default to infinite.

2.33 SD Host

The file ft900_sdhost.h contains the definitions for the SD card device functions in the libft900.a and libft930.a libraries.

2.33.1 Enumeration Type Documentation

2.33.1.1 sdhost_cmd_t
enum sdhost_cmd_t

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDHOST_BUS_CMD</td>
</tr>
<tr>
<td>SDHOST_APP_SPECIFIC_CMD</td>
</tr>
</tbody>
</table>

2.33.1.2 sdhost_response_t
enum sdhost_response_t

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDHOST_RESPONSE_NONE</td>
</tr>
<tr>
<td>SDHOST_RESPONSE_R1</td>
</tr>
</tbody>
</table>
### 2.33.1.3 SDHOST_STATUS

```c
enum SDHOST_STATUS
```

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDHOST_OK</td>
</tr>
<tr>
<td>SDHOST_ERROR</td>
</tr>
<tr>
<td>SDHOST_CARD_INSERTED</td>
</tr>
<tr>
<td>SDHOST_CARD_REMOVED</td>
</tr>
<tr>
<td>SDHOST_INVALID_RESPONSE_TYPE</td>
</tr>
<tr>
<td>SDHOST_CMD_TIMEOUT</td>
</tr>
<tr>
<td>SDHOST_UNUSABLE_CARD</td>
</tr>
<tr>
<td>SDHOST_CMD2_FAILED</td>
</tr>
<tr>
<td>SDHOST_CMD3_FAILED</td>
</tr>
<tr>
<td>SDHOST_CMD8_FAILED</td>
</tr>
<tr>
<td>SDHOST_CMD9_FAILED</td>
</tr>
<tr>
<td>SDHOST_CMD55_FAILED</td>
</tr>
<tr>
<td>SDHOST_ACMD41_FAILED</td>
</tr>
<tr>
<td>SDHOST_CANNOT_ENTER_TRANSFER_STATE</td>
</tr>
<tr>
<td>SDHOST_CANNOT_SET_CARD_BUS_WIDTH</td>
</tr>
<tr>
<td>SDHOST_RESPONSE_ERROR</td>
</tr>
<tr>
<td>SDHOST_WRITE_ERROR</td>
</tr>
<tr>
<td>SDHOST_READ_ERROR</td>
</tr>
</tbody>
</table>
2.33.2 Function Documentation

2.33.2.1 sdhost_abort

SDHOST_STATUS sdhost_abort ( void )
Abort current sdhost operation.

Returns
SDHOST_OK if successful

2.33.2.2 sdhost_card_detect

SDHOST_STATUS sdhost_card_detect ( void )
Check to see if a card is inserted.

Returns
SDHOST_CARD_INSERTED if a card is inserted.
SDHOST_CARD_REMOVED if no card is inserted.

2.33.2.3 sdhost_card_init

SDHOST_STATUS sdhost_card_init ( void )
Identifies and initializes the inserted card. SDHOST can work at baseclock (50Mhz) when the SD card supports it.

Returns
either SDHOST_ERROR or SDHOST_OK

2.33.2.4 sdhost_init

void sdhost_init ( void )
Function initializes SD Host device.

2.33.2.5 sdhost_sys_init

void sdhost_sys_init ( void )
Function used for initializing system registers.

2.33.2.6 sdhost_transfer_data

SDHOST_STATUS sdhost_transfer_data ( uint8_t direction,
            void * buf,
            uint32_t numBytes,
            uint32_t addr
        )
Transfer data to/from SD card.

Parameters

direction  SDHOST_READ or SDHOST_WRITE
buf      address of memory data to be read or written
numBytes size of data to be read or written
addr     address of SD card to write to or read from

Returns
SDHOST_STATUS enum indicating on outcome of operation.

2.34 Datalogger Feature
FT9XX provides several peripherals which may be interfaced to output or storage devices. Developers may use any of these peripherals to output debug or diagnostic information during development. Peripherals attached to storage (SPI flash, SD Card memory, USB Mass Storage, etc.) may be used to store such information, too. However, there are customer applications in which no external storage is available and it becomes impossible to capture and store debug or diagnostic information collected in the field. The datalogger feature uses the on-chip flash in the FT9XX for such storage.

FT90X has 256KB of flash. The flash is organized as a multiple of blocks. Blocks are made up of sectors and sectors in turn are made up of pages. The smallest programmable unit is a page and the smallest erasable unit is a sector. The following table describes the flash geometry in FT90X.

<table>
<thead>
<tr>
<th>Memory Organization</th>
<th>Multiples</th>
<th>Units</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete Flash</td>
<td>4</td>
<td>Blocks</td>
<td>256 KB</td>
</tr>
<tr>
<td>Block</td>
<td>16</td>
<td>Sectors</td>
<td>64 KB</td>
</tr>
<tr>
<td>Sector</td>
<td>16</td>
<td>Page</td>
<td>4 KB</td>
</tr>
<tr>
<td>Page</td>
<td>256</td>
<td>Bytes</td>
<td>256 B</td>
</tr>
</tbody>
</table>

2.34.1 Datalogger Partition
The datalogger partition occupies one sector of the flash and is 4KB in size. As mentioned in FT90X Flash Geometry table Error! Reference source not found., there are 16 pages in one sector. The first and last pages are reserved and the remaining 14 pages are available to the user application for usage. User application refers to the 14 pages via page index 0 to 13.

Once a page has been programmed, it may not be programmed again. In order to overwrite a previously programmed page, the partition has to be erased.
2.34.2 API Cross Reference

It utilizes the following libraries:

`ft900_memctl.h` – FT9xx memory controller driver

The file `ft900_dlog.h` contains the definitions for the datalogger feature functions in the libft900.a and libft930.a libraries.

2.34.3 Variable Documentation

2.34.3.1 __dlog_partition

```c
extern __flash__ uint32_t __dlog_partition[];
```

The global variable `__dlog_partition` has to be referenced in the user application through an extern. The `__dlog_partition` variable is defined in the C runtime for a datalogger application. This value is passed to the `dlog_init()` API function. The `__flash__` attribute informs the compiler that this is a pointer to flash memory.

2.34.4 Function Documentation

2.34.4.1 dlog_init

```c
int dlog_init ( __flash__ uint32_t *flashbuf,
                int    *pgsz,
                int    pages,
            )
```

`dlog_init` must be the first function to be called to initialize the datalogger API. Set `flashbuf` to `__dlog_partition`. On successful return, `pgsz` and `pages` shall be initialized. `Pgsz` indicates the size of the page in flash and `pages` indicates the number of pages available in the partition. In the present API, `pgsz` is fixed to 256 bytes and `pages` is fixed to 14.

`dlog_init` does not keep track of which pages were programmed and which pages remain erased. Such page management is left to the user application.

**Parameters**

- `flashbuf`: Pointer to flash datalog partition
- `pgsz`: Size of page on flash
- `pages`: Number of pages in partition, pg=1...pages

**Returns**

On success a 0, otherwise -1 if partition is invalid.

2.34.4.2 dlog_erase

```c
int dlog_erase ( void )
```

This function is used to erase the flash partition. It then programs the first and last pages with the datalogger signature.

**Returns**

0 when datalog partition was erase, otherwise -1 if datalog library has not been initialized.
2.34.4.3 dlog_read

```c
int dlog_read ( int pg,
                uint32_t *data,
)
```

*dlog_read* is used to read pages from the datalogger partition. *Pg* is an input argument and denotes the user page number to read. Valid values for *pg* are 0 to 13. *data* is an output 32-bit pointer into which page content is transferred to.

**Parameters**

- **pg**  page number, valid range 0..13
- **data** 32-bit pointer to buffer of page size length

**Returns**

On success a 0, otherwise -1 if page or data is invalid.

2.34.4.4 dlog_prog

```c
int dlog_prog ( int pg,
                uint32_t *data,
)
```

*dlog_prog* is used to program pages with user data. *Pg* is an input argument and denotes the user page number to program. Valid values for *pg* are 0 to 13. No check is made if a page was previously programmed. *data* is an input 32-bit pointer containing the information to be programmed.

**Parameters**

- **pg**  page number, valid range 0..13
- **data** 32-bit pointer to buffer of page size length

**Returns**

On success a 0, otherwise -1 if page or data is invalid.

2.35 D2XX Feature

The D2XX interface is a proprietry interface specifically for FTDI devices. A D2XX channel connects two processes; the D2XX application on the USB Host and the user application executing on the FT9xx. Data is exchanged transparently between the peer applications at each end of the channel. It shall relieve the user firmware from dealing with any USB related communication.

D2XX library API calls for the purpose are:

- **D2XX_Init()** – To provide user configuration and initialize the D2XX Solution library.
- **D2XX_Exit()** – To exit D2XX mode and USB device is released to the system.
- **D2XX_Read()** – A read returns data from the D2XX channel. If the channel is empty, zero bytes are returned
- **D2XX_Write()** – A write loads data into the D2XX channel. If the channel is full, the data is not accepted into the channel
D2XX_IOCTL() – Can be used to for remote wakeup or interface related controls

![Figure 2-2 D2XX Solution Context Diagram](image)

### 2.35.1 API Cross Reference

It utilises the following library APIs:

- **ft900_sys.h** – Chip Management
- **ft900_interrupt.h** – Interrupt Management
- **ft900_usbd.h** – USB device API

Additional definitions are taken from

- **ft900_usb.h** – General USB definitions
- **ft900_usb_dfu.h** – USB DFU class definitions

### 2.35.2 Variable Documentation

#### 2.35.2.1 `__d2xx_partition`

```c
extern TD2XX_DeviceConfiguration __pD2XXDefaultConfiguration[];
```

The global variable indicating start of d2xx configuration structure in flash. `__pD2XXDefaultConfiguration` has to be referenced in the user application through an extern. The `__pD2XXDefaultConfiguration` variable is defined in the C runtime for a d2xx firmware application. The default d2xx configuration is available through `ft900_d2xx_default_config.inc` or `ft930_d2xx_default_config.inc` as part of the d2xx project template in the FT9xx toolchain. The user application passes this value to the D2XX_Init() API function.

The configuration can be modified using the FT9xx Toolchain's programmer utility and saved back to the application development project. The configuration structure ends with a 16 bit XOR checksum.

### 2.35.3 Macro Definition Documentation

#### #define D2XX_MAX_INTERFACES (3) [FT90x]

#### #define D2XX_MAX_INTERFACES (7) [FT93x]

The maximum number of D2XX interfaces the D2XX solution for FT9xx can support.

#### #define D2XX_MAX_DESCRIPTOR_STRING_SIZE (128)

The maximum total length of all the four strings used in the string descriptors. Refer TD2XX_DeviceConfiguration for the details on the strings used.
```c
#define D2XX_DEVICEGUID_STRING_SIZE (40)
Double Null terminated ASCII string for Unique device interface GUID in registry format (including Braces and hyphen) for. E.g.: \{2C69C451-55E9-46f0-8E4E-1F30D1E148EE\}.
#define D2XX_API_ERR_BASE (-1)
A non-zero, negative, number base to start the error coding for the Enum ED2XX_ErrorCode.

2.35.4 Structure Documentation

2.35.4.1 TProductDescriptors
Struct to provide the product specific information about D2XX USB device.
Fields:
```c

```c
uint16_t VendorID
```
Vendor ID (assigned by the USB-IF)

```c
uint16_t ProductID
```
Product ID (assigned by the manufacturer)

2.35.4.2 TConfigDescriptors
Struct to provide the configuration descriptor information about D2XX USB device.

Fields:
```c
uint8_t BCDEnable
```
Battery Charge Detection to be enabled or not. 0=disable, 1=enable

```c
uint8_t DFUCapable
```
DFU support. 0=disable, 1=enable

```c
uint8_t SelfPowered
```
Bus or Self Powered Device. 0=disable, 1=enable

```c
uint8_t MaxPower
```
Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2 mA units (i.e., 50 = 100 mA).

```c
uint8_t NumOfD2XXInterfaces
```
Number of D2XX interfaces supported by this USB configuration. Range: 1 to D2XX_MAX_INTERFACES

```c
uint8_t RMWKUPEnable
```
Remote Wakeup capable or not. 0=disable, 1=enable

```c
uint16_t MaxTransferSize
```
The maximum packet size for which transfer happens on each of the D2XX interfaces. Value: 0 or enum values defined in ED2XX_TransferSize. Fill the value to 0 if a d2xx interface is not used (i.e. Indexes >= NumOfD2XXInterfaces)

2.35.4.3 TD2XX_DeviceConfiguration

Fields:
```c
TProductDescriptors ProductDesc
```
Struct to provide the product specific information about D2XX USB device.

```c
TConfigDescriptors ConfigDesc
```
Struct to provide the configuration descriptor information about D2XX USB device.

```c
UInt8_t Strings
```
String configuration section. String 1 – ASCII string detailing the manufacturer.
String 2 – ASCII string detailing the Product.
String 3 – ASCII string for the Serial Number.
String 4 – ASCII string for the DFU Runtime Interface Name.
Note:
All the strings should be preceded with the data length of the string. For e.g.
uint8_t DfuDeviceInterfaceGUID
Double Null terminated ascii string for Unique device interface GUID in registry format (including Braces and hyphen).
For e.g.: \{2C69C451-55E9-46F0-8E4E-1F30D1E148EE\}

uint16_t Checksum
XOR Checksum of the TD2XX_DeviceConfiguration structure

2.35.5 Enumeration Type Documentation

2.35.5.1 ED2XX_ErrorCode
The return values returned by the API functions.

D2XX_ERR_NONE = 0 returns Success
D2XX_ERR_IO = D2XX_API_ERR_BASE in case of IO operation error
D2XX_ERR_MULTI = D2XX_API_ERR_BASE-1 in case of multiple call of the init or exit API
D2XX_ERR_DEVICE = D2XX_API_ERR_BASE-2 in case of Device error
D2XX_ERR_INSUFFICIENT_RESOURCES = D2XX_API_ERR_BASE-3 in case of insufficient memory or resources
D2XX_ERR_INVALID_PARAMETER = D2XX_API_ERR_BASE-4 Invalid Parameter supplied to API function
D2XX_ERR_NOT_SUPPORTED = D2XX_API_ERR_BASE-5 Operation not supported

2.35.5.2 ED2XX_EventCode
The events for which the D2XX API provides callback.

D2XX_EVT_SUSPEND SUSPEND EVENT from USB Host
D2XX_EVT_RESUME RESUME EVENT from USB Host
D2XX_EVT_BUS_RESET BUS RESET EVENT from USB Host
D2XX_EVT_READY D2XX enters Ready state where READ/WRITE requests are processed
D2XX_EVT_UNREADY D2XX exits Ready state (USB disconnected)
D2XX_EVT_DFU_DETACH DFU DETACH Class command from DFU application
D2XX_EVT_TESTMODE D2XX enters Test Mode. Exit is via power cycle
D2XX_EVT_INF_RESET D2XX interface reset command to clear the channel
D2XX_EVT_MAX_CODE

2.35.5.3 ED2XX_IOCTLID
The ioctl id is the unique identifier used by the D2XX IOCTL API to process the ioctl request.

D2XX_IOCTL_SYS_REMOTE_WAKEUP REMOTE WAKEUP to the USB Host
D2XX_IOCTL_INTERFACE_WAKEUP D2XX interface wakeup to be sent for a D2XX interface
D2XX_IOCTL_MAX_ID End of IOCTL ID List

---

Bridgetek Pte Ltd

Copyright © Bridgetek Pte Ltd

Document Feedback
2.35.5.4 ED2XX_TransferSize

The maximum packet size for which transfer happens on the D2XX interface.

- D2XX_XFER_SIZE_32 = 32 32 Bytes
- D2XX_XFER_SIZE_64 = 64 64 Bytes
- D2XX_XFER_SIZE_128 = 128 128 Bytes
- D2XX_XFER_SIZE_256 = 256 256 Bytes
- D2XX_XFER_SIZE_512 = 512 512 Bytes
- D2XX_XFER_SIZE_1024 = 1024 1024 Bytes

2.35.6 Typedef Documentation

2.35.6.1 FD2XX_Callback
typedef void (*FD2XX_Callback)(ED2XX_EventCode eventID, void *ref, void* param1, void* param2);

Callback declaration for user callback functions invoked from D2XX solution.

Parameters

[in] eventID The events for which the D2XX provides callback
[in] ref User application context which is stored and given back during invocation of callback functions on events.
[in] param1 In case of D2XX_EVT_SUSPEND, this PARAM gives whether the RemoteWakeup is enabled or not.

1 – Enabled
0 – Disabled
Based on which the user application can issue a Remote Wakeup to the host device in Suspend mode.

[in] param2 Currently unused.

Returns

void

2.35.7 Function Documentation

2.35.7.1 D2XX_Init

ED2XX_ErrorCode D2XX_Init(TD2XX_DeviceConfiguration *d2xxDeviceConfig, FD2XX_Callback callbackFn, void *ref)

Initialises the D2XX solution library. This function MUST be called prior to any further call to the USB functions.
Important Integration Notes:

1. The D2XX library uses Timer D for scheduling its internal process with a clock prescalar set to 1000. Only the remaining hardware timers – Timer A, Timer B and Timer C are available to the user for the application and these timers have to be initialized for the same prescalar value of 1000.

2. As Timer D is used in the D2XX library, the Timer and the Watchdog hardware block is already enabled in the library through the function call - sys_enable(sys_device_timer_wdt). The API call of sys_enable(sys_device_timer_wdt) cannot be called by the user after D2XX_Init(), as this would affect the D2XX functionality.

Parameters

- **d2xxDeviceConfig** [in] User application custom specific information about the D2XX USB device and its interfaces. This data is used in the construction of device, config and string descriptors.

- **callbackFn** [in] The user application registers its callback function through this param.

- **ref** [in] The user application registers its callback context through this param.

Returns

- D2XX_ERR_NONE if successful.
- D2XX_ERR_INVALID_PARAMETER if invalid values or ranges provided through the d2xxDeviceConfig param.

### 2.35.7.2 D2XX_Exit

```c
void D2XX_Exit(void)
```

The application calls this function to exit D2XX mode. This function cleans up the D2XX solution and USB Driver.

Returns

- void

### 2.35.7.3 D2XX_Read

```c
int32_t D2XX_Read(int32_t interfaceNum, uint8_t *readBuffer, const int32_t length)
```

Performs Read on a D2XX interface.

Parameters

- **interfaceNum** [in] D2XX Interface Number Range: 1..n, n -> Number of D2XX Interfaces configured by the application

- **readBuffer** [in] A pointer to the buffer which the stream data is read into.

- **length** [in] The number of bytes of data to read from D2XX interface buffer.

Returns

- Returns the actual number of bytes read from the D2XX interface.
Zero bytes are returned if the internal D2XX buffer is empty.
D2XX_ERR_INVALID_PARAMETER if invalid values or ranges provided through the
interfaceNum or readBuffer params.
D2XX_ERR_IO if unsuccessful.
D2XX_ERRDEVICE if D2XX is not in the state of processing the request

2.35.7.4 D2XX_Write

int32_t D2XX_Write(int32_t interfaceNum, uint8_t *writeBuffer, const int32_t length)

Performs Write on a D2XX interface.

Parameters
- [in] interfaceNum  D2XX Interface Number Range: 1..n, n -> Number of D2XX Interfaces configured by the application
- [in,out] writeBuffer  A pointer to the buffer to which the stream data is written to.
- [in] length  The number of bytes of data to write from user buffer to the D2XX interface.

Returns
Returns the actual number of bytes written to the D2XX interface.
Zero bytes are returned if the internal D2XX buffer is full.
D2XX_ERR_INVALID_PARAMETER if invalid values or ranges provided through the interfaceNum or writeBuffer params.
D2XX_ERR_IO if unsuccessful.
D2XX_ERRDEVICE if D2XX is not in the state of processing the request

2.35.7.5 D2XX_IOCTL

ED2XX_ErrorCode D2XX_IOCTL(int32_t interfaceNum, int ioctlID, void *param1, void *param2)

The ioctl API is a catch-all that can handle transactions where read and write are not suitable. Typically, this means control data for a D2XX interface or system control of USB device.

Parameters
- [in] interfaceNum  D2XX Interface Number Value: 0 -- System Purpose (e.g. Remote Wakeup) 1..n, n -> Number of D2XX Interfaces configured by the application
- [in] ioctlID  D2XX IOCTL ID Refer to ED2XX_IOCTL documentation
- [in] param1  Additional Parameter that application passes for D2XX_IOCTL_INTERFACE_WAKEUP. It is for set or clear 1 -> Set Wakeup 0 -> Clear Wakeup
Returns

D2XX_ERR_NONE if successful.
D2XX_ERR_INVALID_PARAMETER if invalid values or ranges provided through the interfaceNum or ioctlID params.
D2XX_ERR_DEVICE if D2XX is not in the state of processing the request
D2XX_ERR_NOT_SUPPORTED if any unsupported IOCTL request is made.
3 Header Files

3.1 Hardware Register Definition Files

The following is a list of all hardware register definition files. These are located in the inc/registers directory.

- **ft900_adc_dac_registers.h**  
  ADC/DAC registers
- **ft900_cam_registers.h**  
  Camera/Parallel interface registers
- **ft900_can_registers.h**  
  CANBus registers
- **ft900_ehci_registers.h**  
  EHCI Registers
- **ft900_eth_registers.h**  
  Ethernet registers
- **ft900_flash_registers.h**
- **ft900_gpio_registers.h**  
  General Purpose IO and Pad control registers
- **ft900_i2c_registers.h**  
  I2C registers
- **ft900_i2s_registers.h**  
  I2S registers
- **ft900_interrupt_registers.h**  
  Interrupt management registers
- **ft900_pwm_registers.h**  
  Pulse Width Modulation registers
- **ft900_registers.h**  
  FT90x and FT93x register definitions
- **ft900_rtc_registers.h**  
  Real Time Clock Registers
- **ft900_sdhost_registers.h**  
  SD Host Registers
- **ft900_spi_registers.h**  
  SPI Registers
- **ft900_sys_registers.h**  
  Chip management registers
- **ft900_timer_wdt_registers.h**  
  Timer and Watchdog Registers
- **ft900_uart_registers.h**  
  UART Registers
- **ft900_usbd_registers.h**  
  USBD Registers
- **ft900_usbd_hbw_register.h**  
  High Bandwidth ISO configuration registers
- **ft930_slave_cpu_registers.h**  
  Registers to control the D2XX hardware engine

3.1.1 Using Register Header Files

The register header files can be used to directly access the device registers.

Here are the steps involved:

1. Find the module define in ft900_registers.h
2. Find the register to access in the associated register header file.
3. Decide if any of the constants will be used to help set or clear specific bit fields
3.1.1.1 Example 1 – Read a Register
To read the Chip ID Register in the General System Registers:

```c
uint32_t HIPID_value;
HIPID_value = SYS->HIPID;
```

3.1.1.2 Example 2 – Write to a Register
To set the entire CAN 0 Interrupt enables:

```c
CAN0->CAN_INT_ENABLE |= 0x7F;
```

3.1.1.3 Example 3 – Set and Clear Bits
To bring the CAN 0 module out of reset (set RST to 0):

```c
CAN0->CAN_MODE &= ~MASK_CAN_MODE_RST;
```

To put the CAN 0 back into reset (set RST to 1):

```c
CAN0->CAN_MODE |= MASK_CAN_MODE_RST;
```
### 3.2 API Header Files

This is a list of all the API header files. These are located in the inc directory.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft900.h</td>
<td>FT9xx API (all include files)</td>
</tr>
<tr>
<td>ft900_adc.h</td>
<td>Analogue to Digital Converter</td>
</tr>
<tr>
<td>ft900_asm.h</td>
<td>FT9xx Assembler Macros</td>
</tr>
<tr>
<td>ft900_cam.h</td>
<td>Camera interface</td>
</tr>
<tr>
<td>ft900_can.h</td>
<td>CANBus</td>
</tr>
<tr>
<td>ft900_dac.h</td>
<td>Digital to Analogue Converter</td>
</tr>
<tr>
<td>ft900_delay.h</td>
<td>Delay functions</td>
</tr>
<tr>
<td>ft900_eth.h</td>
<td>Ethernet driver</td>
</tr>
<tr>
<td>ft900_gpio.h</td>
<td>General Purpose I/O and Pad control</td>
</tr>
<tr>
<td>ft900_i2cm.h</td>
<td>I2C Master</td>
</tr>
<tr>
<td>ft900_i2cs.h</td>
<td>I2C Slave</td>
</tr>
<tr>
<td>ft900_i2s.h</td>
<td>I2S Audio</td>
</tr>
<tr>
<td>ft900_interrupt.h</td>
<td>Interrupt management</td>
</tr>
<tr>
<td>ft900_pwm.h</td>
<td>Pulse Width Modulation</td>
</tr>
<tr>
<td>ft900_pwm_pcm.h</td>
<td>PWM Audio</td>
</tr>
<tr>
<td>ft900_rtc.h</td>
<td>Real Time Clock</td>
</tr>
<tr>
<td>ft900_sdhost.h</td>
<td>SD Host</td>
</tr>
<tr>
<td>ft900_spi.h</td>
<td>SPI</td>
</tr>
<tr>
<td>ft900_startup_dfu.h</td>
<td>Startup DFU Feature</td>
</tr>
<tr>
<td>ft900_sys.h</td>
<td>Chip management</td>
</tr>
<tr>
<td>ft900_timers.h</td>
<td>Timers</td>
</tr>
<tr>
<td>ft900_uart_simple.h</td>
<td>UART</td>
</tr>
<tr>
<td>ft900_usb.d</td>
<td>USB Device API</td>
</tr>
<tr>
<td>ft900_usb_dfu.h</td>
<td>DFU device for USB device stack API</td>
</tr>
<tr>
<td>ft900_usb_hbw.h</td>
<td>USB Device High Bandwidth Isochronous IN support API on FT90x Rev C.</td>
</tr>
<tr>
<td>ft900_usb_rndis.h</td>
<td>RNDIS Device for USB device stack API</td>
</tr>
<tr>
<td>ft900_usbh.h</td>
<td>USB host stack API</td>
</tr>
<tr>
<td>ft900_usbh_boms.h</td>
<td>BOMS devices on USB host stack API</td>
</tr>
<tr>
<td>ft900_usbh_cdcacm.h</td>
<td>CDC ACM devices on USB host stack API</td>
</tr>
<tr>
<td>ft900_usbh_hid.h</td>
<td>HID devices on USB host stack API</td>
</tr>
</tbody>
</table>
3.3 Additional Header Files
This list contains all additional header files that are not directly part of the API but provide additional definitions for the API and applications. They are located in the inc directory.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft900_usb.h</td>
<td>USB definitions</td>
</tr>
<tr>
<td>ft900_usb_boms.h</td>
<td>USB BOMS class definitions</td>
</tr>
<tr>
<td>ft900_usb_cdc.h</td>
<td>USB CDC class USB definitions</td>
</tr>
<tr>
<td>ft900_usb_dfu.h</td>
<td>USB DFU class definitions</td>
</tr>
<tr>
<td>ft900_usb_hid.h</td>
<td>USB HID class definitions</td>
</tr>
<tr>
<td>ft900_usb_aoa.h</td>
<td>USB AOA class definitions</td>
</tr>
<tr>
<td>ft900_usb_rndis.h</td>
<td>USB RNDIS class definitions</td>
</tr>
<tr>
<td>ft900_usb_uvc.h</td>
<td>USB UVC class definitions</td>
</tr>
<tr>
<td>ft900_usbh_aoa.h</td>
<td>AOA devices on USB host stack API</td>
</tr>
<tr>
<td>ft900_usbhx.h</td>
<td>USB host API extensions</td>
</tr>
<tr>
<td>ft900_wdt.h</td>
<td>Watchdog Timer</td>
</tr>
<tr>
<td>ftd2xx_api.h</td>
<td>D2XX Solution API for FT9xx</td>
</tr>
</tbody>
</table>
4 Contact Information

Head Quarters – Singapore

Bridgetek Pte Ltd
178 Paya Lebar Road, #07-03
Singapore 409030
Tel: +65 6547 4827
Fax: +65 6841 6071

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office – Taipei, Taiwan

Bridgetek Pte Ltd, Taiwan Branch
2 Floor, No. 516, Sec. 1, Nei Hu Road, Nei Hu District
Taipei 114
Taiwan, R.O.C.
Tel: +886 (2) 8797 5691
Fax: +886 (2) 8751 9737

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office - Glasgow, United Kingdom

Bridgetek Pte. Ltd.
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales.emea@brtchip.com
E-mail (Support) support.emea@brtchip.com

Branch Office – Vietnam

Bridgetek VietNam Company Limited
Lutaco Tower Building, 5th Floor, 173A Nguyen Van Troi,
Ward 11, Phu Nhuan District,
Ho Chi Minh City, Vietnam
Tel : 08 38453222
Fax : 08 38455222

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Web Site
http://brtchip.com/

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Bridgetek Pte Ltd (BRTChip) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested Bridgetek devices and other materials) is provided for reference only. While Bridgetek has taken care to assure it is accurate, this information is subject to customer confirmation, and Bridgetek disclaims all liability for system designs and for any applications assistance provided by Bridgetek. Use of Bridgetek devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold harmless Bridgetek from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Bridgetek Pte Ltd, 178 Paya Lebar Road, #07-03, Singapore 409030. Singapore Registered Company Number: 201542387H.
Appendix A – References

Document References

FT900/FT901/FT902/FT903 Datasheet
FT905/FT906/FT907/FT908 Datasheet
FT930/FT931/FT932/FT933 Datasheet

USB-IF Device Firmware Upgrade 1.1 Specification

USB-IF Class definitions for Communication Devices 1.2
https://www.usb.org/document-library/class-definitions-communication-devices-12
https://www.usb.org/sites/default/files/CDC1.2_WMC1.1_012011.zip

USB-IF Device Class Definitions for HID 1.11
https://www.usb.org/document-library/device-class-definition-hid-111

USB-IF Mass Storage Bulk Only

Android Open Accessory Specification
https://source.android.com/devices/accessories/aoa2.html

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analogue to Digital Converter</td>
</tr>
<tr>
<td>AOA</td>
<td>Android Open Accessory</td>
</tr>
<tr>
<td>ARP</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>BOMS</td>
<td>Bulk Only Mass Storage</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>CDC</td>
<td>Communication Device Class</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analogue Converter</td>
</tr>
<tr>
<td>DFU</td>
<td>Device Firmware Upgrade</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electronically Erasable PROgrammable Memory</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose I/O</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>HID</td>
<td>Human Interface Device</td>
</tr>
<tr>
<td>I²C</td>
<td>Inter-IC</td>
</tr>
<tr>
<td>I²S</td>
<td>Inter-IC Sound</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Messaging Protocol</td>
</tr>
<tr>
<td>MDI-X</td>
<td>Medium Dependent Interface Crossover</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation</td>
</tr>
<tr>
<td>RNDIS</td>
<td>Remote Network Driver Interface Specification</td>
</tr>
<tr>
<td>RTC</td>
<td>Real Time Clock</td>
</tr>
<tr>
<td>SD</td>
<td>Secure Digital</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>WDT</td>
<td>Watchdog Timer</td>
</tr>
</tbody>
</table>
Appendix B – List of Tables & Figures

List of Tables
Table 1- Precompiled Libraries released with FT9XX Toolchain .................................................. 8
Table 2- FT90X Pin Mapping ........................................................................................................... 18
Table 3- FT93X Pin Mapping ........................................................................................................... 20
Table 4- Ethernet buffer format ..................................................................................................... 59
Table 5- UART Baudrate table ........................................................................................................ 65
Table 6- I2CM Status Mask ........................................................................................................... 75
Table 7- I2CM FIFO interrupt Mask ............................................................................................... 76
Table 8- I2CS Status Mask ............................................................................................................. 79
Table 9- I2S Interrupt Enable Mask ............................................................................................... 85
Table 10- SPI Status flags .............................................................................................................. 92
Table 11- SPIIM Options ................................................................................................................ 93
Table 12- CAN mode and message filters ...................................................................................... 99
Table 13- CAN Status Bit Mask .................................................................................................... 101
Table 14- CAN Error Code Mask .................................................................................................. 102
Table 15- CAN Interrupt Enable Mask .......................................................................................... 103
Table 16 – FT90X Flash Geometry ................................................................................................. 206

List of Figures
Figure 1-1 FT90X Peripherals Driver Support ............................................................................... 7
Figure 1-2 FT93X Interface Driver Support ................................................................................... 7
Figure 2-1 Datalogger Partition ..................................................................................................... 206
Figure 2-2 D2XX Solution Context Diagram ................................................................................. 209
# Appendix C – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>2015-06-29</td>
</tr>
<tr>
<td>1.1</td>
<td>Updated to reflect changes in USB Host API</td>
<td>2015-09-14</td>
</tr>
<tr>
<td>1.2</td>
<td>Added USBH AOA API, Datalogger and D2XX API</td>
<td>2016-02-25</td>
</tr>
<tr>
<td>1.3</td>
<td>Updated USBD, STARTUP_DFU sections and Added ‘FT devices on USB host stack’ API</td>
<td>2016-09-20</td>
</tr>
<tr>
<td>1.4</td>
<td>Updated release Migration of the product from FTDI to Bridgetek name – logo changed, copyright changed, contact information changed</td>
<td>2017-03-09</td>
</tr>
<tr>
<td>1.5</td>
<td>Updated D2XX_Init() call with the integration notes about the usage of Timer D inside the library.</td>
<td>2017-07-05</td>
</tr>
<tr>
<td>1.6</td>
<td>Updated document for - FT900C porting. Following topics are updated – SD Host, CAN, RTC, SPI, I2S, I2C, UART, USBD_HW; RTC on the compatibility for FT900 rev B.</td>
<td>2018-01-19</td>
</tr>
<tr>
<td>1.7</td>
<td>Updated the document for ADC, UART and USBD</td>
<td>2018-11-14</td>
</tr>
</tbody>
</table>