Color Controller

Generated by Doxygen 1.8.9.1

Thu Aug 20 2015 08:22:39

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

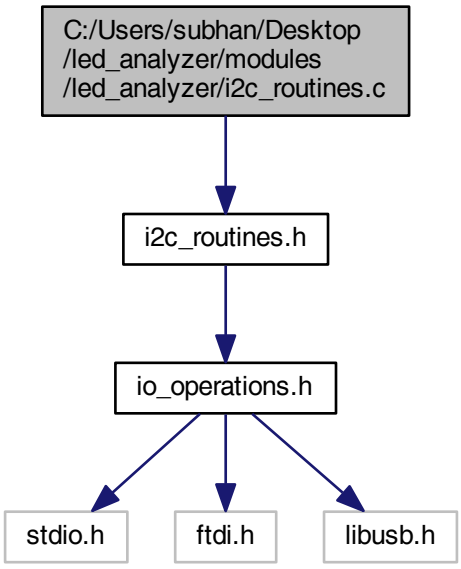
File Documentation

2.1 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/i2c_routines.c File Reference

Software I2C Functions for the FTDI 2232H Chip.

```
#include "i2c_routines.h"
```

Include dependency graph for i2c_routines.c:



Functions

- void i2c_startCond ()

  sends a start condition on all 16 i2c-busses.
- void i2c_stopCond ()

  sends a stop condition on all 16 i2c-busses.

- int i2c_write8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength)

  i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.
- int i2c_write8_x (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned int uiX)

  i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.
- int i2c_read8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned char *aucRecBuffer, unsigned char ucRecLength)

  i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in aucRecBuffer.
- int i2c_read16 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned short *ausReadBuffer, unsigned char ucRecLength)

  i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in an adequate buffer.
- int i2c_read72 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned char *aucStatusRegister, unsigned short *ausReadBuffer1, unsigned short *aus↩ ReadBuffer2, unsigned short *ausReadBuffer3, unsigned short *ausReadBuffer4, unsigned char ucRec↩ Length)

  i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in adequates buffer.
- void i2c_clock (unsigned long ulDataToSend)

  triggers a clock cycle on all clock lines, while sendindg out data on the data lines.
- void i2c_clockInput (unsigned long ulDataToSend)

  triggers a clock cycle on the clock lines while expecting data from the slave on the data lines.
- void i2c_giveAck ()

  master gives an acknowledge on all data lines.
- void i2c_clock_forACK (unsigned long ulDataToSend)

  clocks the acknowledge bit given by the slave.
- void i2c_getAck ()

  expects and clocks an acknowledge bit given by the slave.

2.1.1 Detailed Description

Software I2C Functions for the FTDI 2232H Chip.

i2c_routines is a simple library which provides basic i2c-functionality. Functions include sending 1 or more bytes, and reading 1 Byte / 2 Bytes. The structure of the buffers which will be sent consists of [address - register - data]. This i2c-library can be used for simple i2c-slaves which do not have the ability of clock stretching.

Warning

clock stretching and multi-master-mode is not supported

2.1.2 Function Documentation

2.1.2.1 void i2c_clock (unsigned long *ulDataToSend*)

triggers a clock cycle on all clock lines, while sendindg out data on the data lines.

Parameters

ulDataToSend	data which is going to be clocked on all lines set as output

2.1.2.2 void i2c_clock_forACK (unsigned long *ulDataToSend*)

clocks the acknowledge bit given by the slave.

Parameters

ulDataToSend	data which is going to be clocked on lines set as output

2.1.2.3 void i2c_clockInput (unsigned long *ulDataToSend*)

triggers a clock cycle on the clock lines while expecting data from the slave on the data lines.

ulDataToSend is the value which is going to be written to all pins set as output, if no pin is set as output, nothing happens. All pins which are set as input will capute their value on the negative clock edge.

Parameters

ulDataToSend	data which is going to be clocked on all lines set as output

2.1.2.4 int i2c_read16 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucSendBuffer*, unsigned char *ucLength*, unsigned short * *ausReadBuffer*, unsigned char *ucRecLength*)

i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in an adequate buffer.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can read 2 bytes from an i2c-slave.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains slave address and register to read from
ucLength	sizeof aucSendbuffer in bytes
ausReadBuffer	stores the information read back from the slaves
ausReadBuffer	as we have 16 i2c-busses ausReadBuffer must be able to hold at least 16 elements.
ucRecLength	number of expected bytes to read from the slaves

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.1.2.5 int i2c_read72 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucSendBuffer*, unsigned char *ucLength*, unsigned char * *aucStatusRegister*, unsigned short * *ausReadBuffer1*, unsigned short * *ausReadBuffer2*, unsigned short * *ausReadBuffer3*, unsigned short * *ausReadBuffer4*, unsigned char *ucRecLength*)

i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in adequates buffer.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can read 4 times 2 bytes from an i2c-slave. Internally it reads one more Byte (the status register) in order to check if conversions have already completed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains slave address and register to read from
ucLength	sizeof aucSendbuffer in bytes
aucStatus↩ Register	stores the content of the status register read back from the slaves
ausReadBuffer1	stores the first 16 Bit information read back from the slaves
ausReadBuffer2	stores the second 16 Bit information read back from the slaves
ausReadBuffer3	stores the third 16 Bit information read back from the slaves
ausReadBuffer4	stores the fourth 16 Bit information read back from the slaves
ucRecLength	number of expected bytes to read from the slaves

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.1.2.6 int i2c_read8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucSendBuffer*, unsigned char *ucLength*, unsigned char * *aucRecBuffer*, unsigned char *ucRecLength*)

i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in aucRecBuffer.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can read one byte from an i2c-slave.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains address and register to read from
ucLength	sizeof aucSendbuffer in bytes
aucRecBuffer	buffer which will store the information read back from the slaves
aucRecBuffer	as we have 16 i2c-busses aucRecBuffer must be able to hold at least 16 bytes.
ucRecLength	sizeof aucRecBuffer in bytes

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.1.2.7 int i2c_write8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucSendBuffer*, unsigned char *ucLength*)

i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can send one byte to a 8 Bit Register of a i2c-slave. Though the name is i2c_write8, the function can send out as many bytes as wanted. The number of bytes to be sent can be passed in the parameter ucLength.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains address, register and data
ucLength	sizeof aucSendbuffer in bytes

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.1.2.8 int i2c_write8_x (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucSendBuffer,* unsigned char *ucLength,* unsigned int *uiX*)

i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.

Sends the amount of bytes specified in ucLength over one of the 16 i2c-busses. The number of the i2c-bus which shall send the data will be given in uiX, which ranges from 0 ... 15.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains address, register and data
ucLength	sizeof aucSendbuffer in bytes
uiX	number of i2c-bus which should send the data (0 ... 15)

Returns

0 if succesful, errorcode if not

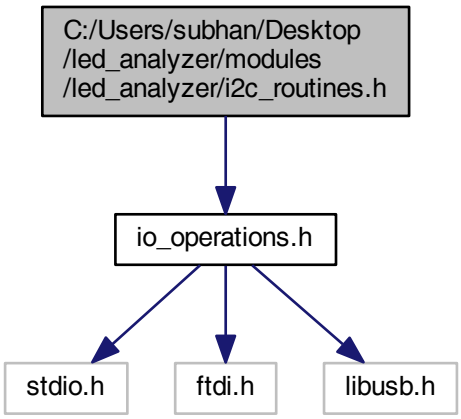
- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.2 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/i2c_routines.h File Reference

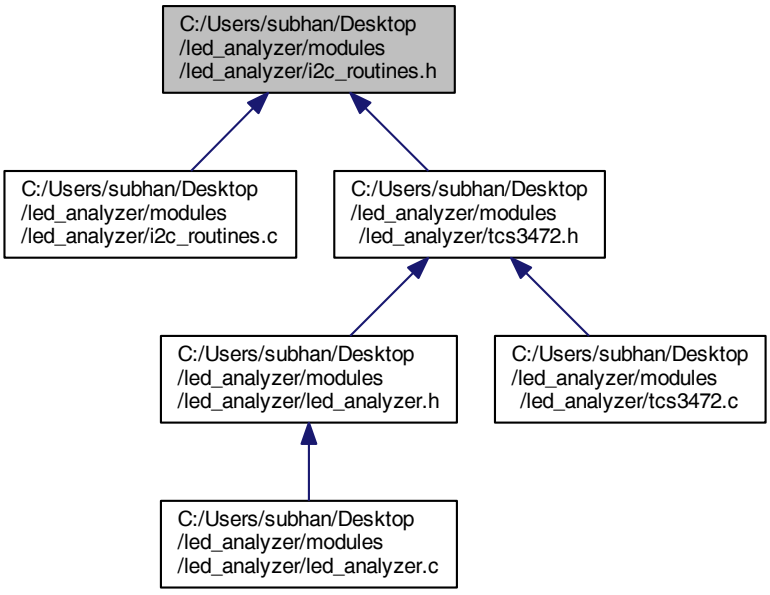
Software I2C Functions for the FTDI 2232H Chip (header)

```
#include "io_operations.h"
```

Include dependency graph for i2c_routines.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define SDA_WRITE 0x0
- #define SDA_READ 0x55555555
- #define SCL 0xAAAAAAAA

- #define SDA_0_OUTPUT 0x55
- #define SDA_0_INPUT 0x00
- #define SDA_1_OUTPUT 0x5500
- #define SDA_1_INPUT 0x00
- #define SDA_2_OUTPUT 0x550000
- #define SDA_2_INPUT 0x00
- #define SDA_3_OUTPUT 0x55000000
- #define SDA_3_INPUT 0x00

Functions

- void i2c_startCond ()

  sends a start condition on all 16 i2c-busses.
- void i2c_stopCond ()

  sends a stop condition on all 16 i2c-busses.
- void i2c_clock (unsigned long ulDataToSend)

  triggers a clock cycle on all clock lines, while sendindg out data on the data lines.
- void i2c_clockInput (unsigned long ulDataToSend)

  triggers a clock cycle on the clock lines while expecting data from the slave on the data lines.
- void i2c_giveAck ()

  master gives an acknowledge on all data lines.
- void i2c_clock_forACK (unsigned long ulDataToSend)

  clocks the acknowledge bit given by the slave.
- void i2c_getAck ()

  expects and clocks an acknowledge bit given by the slave.
- int i2c_read16 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned short *ausReadBuffer, unsigned char ucRecLength)

  i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in an adequate buffer.
- int i2c_read72 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned char *aucStatusRegister, unsigned short *ausReadBuffer1, unsigned short *aus↩ ReadBuffer2, unsigned short *ausReadBuffer3, unsigned short *ausReadBuffer4, unsigned char ucRec↩ Length)

  i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in adequates buffer.
- int i2c_read8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned char *aucRecBuffer, unsigned char ucRecLength)

  i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in aucRecBuffer.
- int i2c_write8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength)

  i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.
- int i2c_write8_x (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucSendBuffer, unsigned char ucLength, unsigned int uiX)

  i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.

2.2.1 Detailed Description

Software I2C Functions for the FTDI 2232H Chip (header)

i2c_routines is simple library which provides basic i2c-functionality. Functions include sending 1 or more bytes, and reading 1 Byte / 2 Bytes. The structure of the buffers which will be sent consists of [address - register - data]. This i2c-library can be used for simple i2c-slaves which do not have the ability of clock stretching.

Warning

clock stretching and multi-master-mode is not supported

2.2.2 Macro Definition Documentation

2.2.2.1 #define SCL 0xAAAAAAAA

mask which maps to all clock lines of the ftdi 2232h chip

2.2.2.2 #define SDA_0_INPUT 0x00

mask which sets all data lines of channel AD (lowbyte) as input

2.2.2.3 #define SDA_0_OUTPUT 0x55

mask which sets all data lines of channel AD (lowbyte) as output

2.2.2.4 #define SDA_1_INPUT 0x00

mask which sets all data lines of channel AC (highbyte) as input

2.2.2.5 #define SDA_1_OUTPUT 0x5500

mask which sets all data lines of channel AC (highbyte) as output

2.2.2.6 #define SDA_2_INPUT 0x00

mask which sets all data lines of channel BD (lowbyte) as output

2.2.2.7 #define SDA_2_OUTPUT 0x550000

mask which sets all data lines of channel BD (lowbyte) as output

2.2.2.8 #define SDA_3_INPUT 0x00

mask which sets all data lines of channel BC (highbyte) as input

2.2.2.9 #define SDA_3_OUTPUT 0x55000000

mask which sets all data lines of channel BC (highbyte) as output

2.2.2.10 #define SDA_READ 0x55555555

MSB after 7 address bits should be high to trigger a i2c-read access

2.2.2.11 #define SDA_WRITE 0x0

MSB after 7 address bits should be low to trigger a i2c-write access

2.2.3 Function Documentation

2.2.3.1 void i2c_clock (unsigned long *ulDataToSend*)

triggers a clock cycle on all clock lines, while sendindg out data on the data lines.

Parameters

ulDataToSend	data which is going to be clocked on all lines set as output

2.2.3.2 void i2c_clock_forACK (unsigned long *ulDataToSend*)

clocks the acknowledge bit given by the slave.

Parameters

ulDataToSend	data which is going to be clocked on lines set as output

2.2.3.3 void i2c_clockInput (unsigned long *ulDataToSend*)

triggers a clock cycle on the clock lines while expecting data from the slave on the data lines.

ulDataToSend is the value which is going to be written to all pins set as output, if no pin is set as output, nothing happens. All pins which are set as input will capute their value on the negative clock edge.

Parameters

ulDataToSend	data which is going to be clocked on all lines set as output

2.2.3.4 int i2c_read16 (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucSendBuffer,* unsigned char *ucLength,* unsigned short * *ausReadBuffer,* unsigned char *ucRecLength*)

i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in an adequate buffer.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can read 2 bytes from an i2c-slave.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains slave address and register to read from
ucLength	sizeof aucSendbuffer in bytes
ausReadBuffer	stores the information read back from the slaves
ausReadBuffer	as we have 16 i2c-busses ausReadBuffer must be able to hold at least 16 elements.
ucRecLength	number of expected bytes to read from the slaves

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.2.3.5 int i2c_read72 (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucSendBuffer,* unsigned char *ucLength,* unsigned char * *aucStatusRegister,* unsigned short * *ausReadBuffer1,* unsigned short * *ausReadBuffer2,* unsigned short * *ausReadBuffer3,* unsigned short * *ausReadBuffer4,* unsigned char *ucRecLength*)

i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in adequates buffer.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can read 4 times 2 bytes from an i2c-slave. Internally it reads one more Byte (the status register) in order to check if conversions have already completed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains slave address and register to read from
ucLength	sizeof aucSendbuffer in bytes
aucStatus↩ Register	stores the content of the status register read back from the slaves
ausReadBuffer1	stores the first 16 Bit information read back from the slaves
ausReadBuffer2	stores the second 16 Bit information read back from the slaves
ausReadBuffer3	stores the third 16 Bit information read back from the slaves
ausReadBuffer4	stores the fourth 16 Bit information read back from the slaves
ucRecLength	number of expected bytes to read from the slaves

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.2.3.6 int i2c_read8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucSendBuffer*, unsigned char *ucLength*, unsigned char * *aucRecBuffer*, unsigned char *ucRecLength*)

i2c-function reads the slaves connected to all 16 i2c-busses and stores the information in aucRecBuffer.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can read one byte from an i2c-slave.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains address and register to read from
ucLength	sizeof aucSendbuffer in bytes
aucRecBuffer	buffer which will store the information read back from the slaves
aucRecBuffer	as we have 16 i2c-busses aucRecBuffer must be able to hold at least 16 bytes.
ucRecLength	sizeof aucRecBuffer in bytes

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.2.3.7 int i2c_write8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucSendBuffer*, unsigned char *ucLength*)

i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.

ftdiA and ftdiB represent Channel A and Channel B of a ftdi device and each of these channels has 8 i2c-busses. This function can send one byte to a 8 Bit Register of a i2c-slave. Though the name is i2c_write8, the function can send out as many bytes as wanted. The number of bytes to be sent can be passed in the parameter ucLength.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains address, register and data
ucLength	sizeof aucSendbuffer in bytes

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.2.3.8 int i2c_write8_x (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucSendBuffer,* unsigned char *ucLength,* unsigned int *uiX*)

i2c-function sends the content of aucSendBuffer on all 16 i2c-busses.

Sends the amount of bytes specified in ucLength over one of the 16 i2c-busses. The number of the i2c-bus which shall send the data will be given in uiX, which ranges from 0 ... 15.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucSendBuffer	pointer to the buffer which contains address, register and data
ucLength	sizeof aucSendbuffer in bytes
uiX	number of i2c-bus which should send the data (0 ... 15)

Returns

0 if succesful, errorcode if not

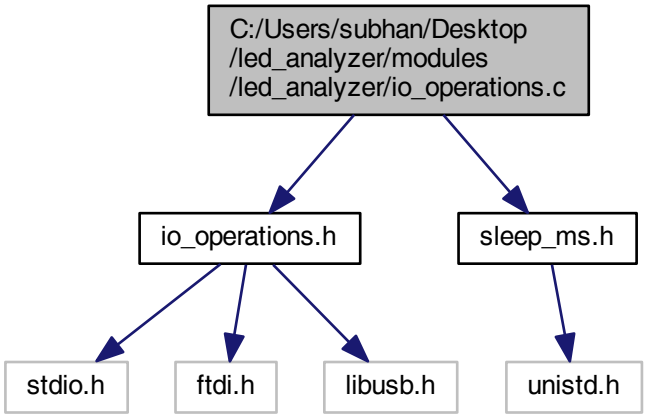
- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.3 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/io_operations.c File Reference

provides functions to manipulate ftdi 2232h's I/O-Pins

```
#include "io_operations.h"
#include "sleep_ms.h"
```

Include dependency graph for io_operations.c:



Functions

- int writeOutputs (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, const unsigned long ulOutput)

  writes a value to the ftdi 2232h output pins.
- int readInputs (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, const unsigned char *readBack)

  reads the input pins of both ftdi channels.
- void process_pins (unsigned long ulIOMask, unsigned long ulOutput)

  stores a ftdi write command in a global buffer for later sending.
- void process_pins_databack (unsigned long ulIOMask, unsigned long ulOutput)

  stores a ftdi write command in a global buffer for later sending.
- int send_package_write8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)

  sends the content of the global buffers to the ftdi chip.
- int send_package_read8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucReadBuffer, unsigned char ucReadBufferLength)

  sends the content of the global buffers to the ftdi chip.
- int send_package_read16 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausReadBuffer, unsigned char ucReadBufferLength)

  sends the content of the global buffers to the ftdi chip.
- int send_package_read72 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucReadBuffer, unsigned short *ausReadBuffer1, unsigned short *ausReadBuffer2, unsigned short *ausReadBuffer3, unsigned short *ausReadBuffer4, unsigned char ucReadBufferLength)

  sends the content of the global buffers to the ftdi chip.

Variables

- unsigned int indexA = 0
- unsigned int indexB = 0
- unsigned int readIndexA = 0
- unsigned int readIndexB = 0
- unsigned char aucBufferA [4096]
- unsigned char aucBufferB [4096]

2.3.1 Detailed Description

provides functions to manipulate ftdi 2232h's I/O-Pins

Once the ftdi2232h is set into BITMODE_MPSSE simple USB commands can be sent to it in order to manipulate its input and output pins. Special commands (for example found in AN_108) can be used to set the 32 GPIO Pins of the ftdi device as either input or output, and once done, values can be assigned to the output pins and data can be read back from the input pins. These functions will be used to provide software i2c functionality.

2.3.2 Function Documentation

2.3.2.1 void process_pins (unsigned long *ullOMask,* unsigned long *ulOutput*)

stores a ftdi write command in a global buffer for later sending.

This function gets called repeatedly by i2c functions. It stores the commands in global Buffers (aucBufferA and aucBufferB). The commands consist of a mask which determines which pins are configured as output and input plus the actual output value to be written to the pins. All stored commands can be sent by the send_package_xx functions which form the software i2c protocol.

Parameters

in	*ullOMask*	input / output mask to set pin functionality
in	*ulOutput*	value to be assigned to pins set as output

2.3.2.2 void process_pins_databack (unsigned long *ullOMask,* unsigned long *ulOutput*)

stores a ftdi write command in a global buffer for later sending.

This function gets called repeatedly by i2c functions. It stores the commands in global Buffers (aucBufferA and aucBufferB). The commands consist of a mask which determines which pins are set as input and output and and output value which will be written to the pins set as output. All stored commands can be sent by the send_↩ package_xx functions which form the software i2c protocol.

Parameters

in	*ullOMask*	input / output mask to set pin direction
in	*ulOutput*	value to be assigned to pins set as output

2.3.2.3 int readInputs (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* const unsigned char * *readBack*)

reads the input pins of both ftdi channels.

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*readBack*	contains the bytes read back from the input pins

Returns

>0 : number of bytes written to the chip
<0 : USB functions failed

2.3.2.4 int send_package_read16 (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausReadBuffer,* unsigned char *ucReadBufferLength*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip. Furthermore it reads back the data of pins which were configured as input. The function returns a value which equals the amount of read back bytes. The parameter ausReadbuffer will be used for storing 16 read back unsigned short int values of 16 sensors

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*ausReadBuffer*	pointer to array of unsigned short values
in	*ucReadBufferLength*	number of elements to be stored in ausReadbuffer

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.3.2.5 int send_package_read72 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucReadBuffer*, unsigned short * *ausReadBuffer1*, unsigned short * *ausReadBuffer2*, unsigned short * *ausReadBuffer3*, unsigned short * *ausReadBuffer4*, unsigned char *ucReadBufferLength*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip. Furthermore it reads back the data of pins which were configured as input. The function returns a value which equals the amount of read back bytes. The parameter ausReadbuffer will be used for storing 16 read back unsigned short int values of 16 sensors

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*aucReadBuffer*	pointer to array of unsigned char values
in,out	*ausReadBuffer1*	pointer to array of unsigned short values
in,out	*ausReadBuffer2*	pointer to array of unsigned short values
in,out	*ausReadBuffer3*	pointer to array of unsigned short values
in,out	*ausReadBuffer4*	pointer to array of unsigned short values
in	*ucReadBufferLength*	number of elements to be stored in ausReadbuffer

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.3.2.6 int send_package_read8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucReadBuffer*, unsigned char *ucReadBufferLength*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip. Furthermore it reads back the data of pins which were configured as input. The function returns a value which equals the amount of read back bytes. The parameter aucReadbuffer will be used for storing 16 read back unsigned char values

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*aucReadBuffer*	pointer to array of unsigned char values
in	*ucReadBuffer↩ Length*	number of elements to be stored in aucReadbuffer

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.3.2.7 int send_package_write8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip Furthermore it reads back the data of pins which were configured as input. In case of i2c these read back pins can be acknowledge bits or data send back by the device.

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.3.2.8 int writeOutputs (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, const unsigned long *ulOutput*)

writes a value to the ftdi 2232h output pins.

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in	*ulOutput*	a 32 Bit value to be written to the ftdi pins
in	*ulOutput*	Bit0 will be assigned to AD0, Bit31 to BC7

Returns

>0 : number of bytes written to the chip
<0 : USB functions failed

2.3.3 Variable Documentation

2.3.3.1 unsigned char aucBufferA[4096]

global Buffer stores the commands for channel A

2.3.3.2 unsigned char aucBufferB[4096]

global Buffer stores the commands for channel B

2.3.3.3 unsigned int indexA = 0

global arrayindex for Channel A, it marks the current index of aucBufferA

2.3.3.4 unsigned int indexB = 0

global arrayindex for Channel B, it marks the current index of aucBufferB

2.3.3.5 unsigned int readIndexA = 0

global readIndex for Channel A, incremented everytime a byte is expected to be read back from Channel A

2.3.3.6 unsigned int readIndexB = 0

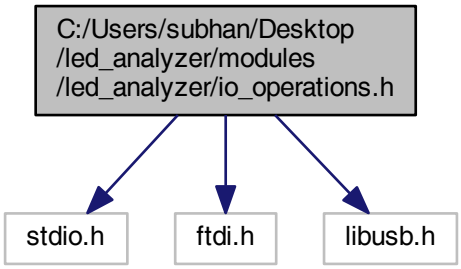
global readIndex for Channel B, incremented everytime a byte is expected to be read back from Channel B

2.4 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/io_operations.h File Reference

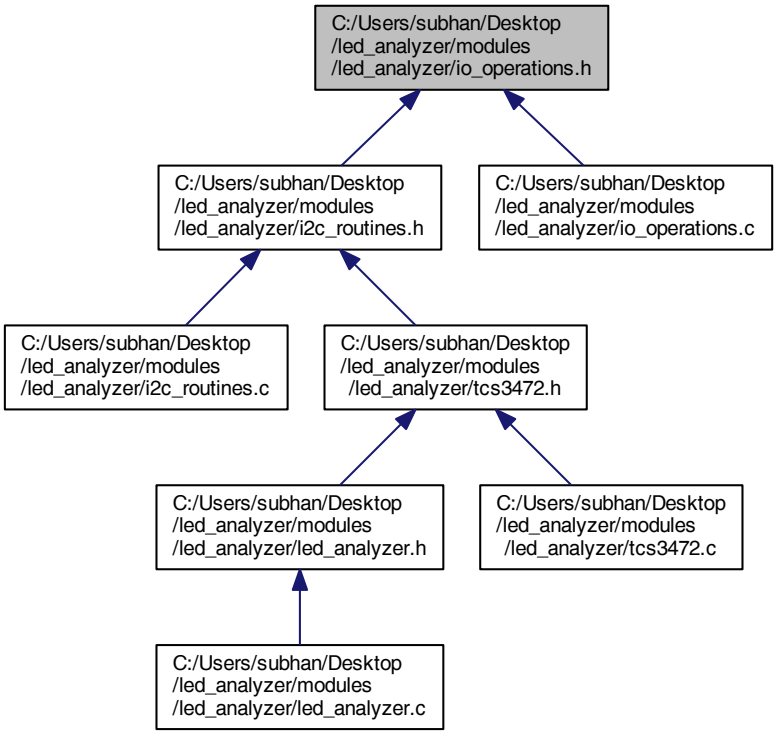
provides functions to manipulate ftdi 2232h's I/O-Pins (header)

```
#include <stdio.h>
#include "ftdi.h"
#include "libusb.h"
```

Include dependency graph for io_operations.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define W_LOWBYTE 0x80
- #define W_HIGHBYTE 0x82
- #define R_LOWBYTE 0x81
- #define R_HIGHBYTE 0x83

- #define MASK_ALOW 0x000000FF
- #define MASK_AHIGH 0x0000FF00
- #define MASK_BLOW 0x00FF0000
- #define MASK_BHIGH 0xFF000000
- #define OUTPUT 0xFF
- #define MYINPUT 0x00
- #define WRITE_ERR_CH_A -1
- #define WRITE_ERR_CH_B -2
- #define READ_ERR_CH_A -3
- #define READ_ERR_CH_B -4
- #define ERR_INCORRECT_AMOUNT -5

Functions

- int writeOutputs (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, const unsigned long ulOutput)

  writes a value to the ftdi 2232h output pins.
- int readInputs (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, const unsigned char *readBack)

  reads the input pins of both ftdi channels.
- void process_pins (unsigned long ulIOMask, unsigned long ulOutput)

  stores a ftdi write command in a global buffer for later sending.
- void process_pins_databack (unsigned long ulIOMask, unsigned long ulOutput)

  stores a ftdi write command in a global buffer for later sending.
- int send_package_write8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)

  sends the content of the global buffers to the ftdi chip.
- int send_package_read8 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucReadBuffer, unsigned char ucReadBufferLength)

  sends the content of the global buffers to the ftdi chip.
- int send_package_read16 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausRead↩ Buffer, unsigned char ucReadBufferLength)

  sends the content of the global buffers to the ftdi chip.
- int send_package_read72 (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucRead↩ Buffer, unsigned short *ausReadBuffer1, unsigned short *ausReadBuffer2, unsigned short *ausRead↩ Buffer3, unsigned short *ausReadBuffer4, unsigned char ucReadBufferLength)

  sends the content of the global buffers to the ftdi chip.

2.4.1 Detailed Description

provides functions to manipulate ftdi 2232h's I/O-Pins (header)

Once the ftdi2232h is set into BITMODE_MPSSE simple USB commands can be sent to it in order to manipulate its input and output pins. Special commands (for example found in AN_108) can be used to set the 32 GPIO Pins of the ftdi device as either input or output, and once done, values can be assigned to the output pins and data can be read back from the input pins. These functions will be used to provide software i2c functionality.

2.4.2 Macro Definition Documentation

2.4.2.1 #define ERR_INCORRECT_AMOUNT -5

Error - received a different amount of bytes than expected

2.4.2.2 #define MASK_AHIGH 0x0000FF00

Mask for higbyte of Channel A

2.4.2.3 #define MASK_ALOW 0x000000FF

Mask for lowbyte of channel A

2.4.2.4 #define MASK_BHIGH 0xFF000000

Mask for highbyte of channel B

2.4.2.5 #define MASK_BLOW 0x00FF0000

Mask for lowbyte of channel B

2.4.2.6 #define MYINPUT 0x00

Set all Bits as input

2.4.2.7 #define OUTPUT 0xFF

Set 8 Bits as output

2.4.2.8 #define R_HIGHBYTE 0x83

Read command for highbyte (AC, BC)

2.4.2.9 #define R_LOWBYTE 0x81

Read command for lowbyte (AD, BD)

2.4.2.10 #define READ_ERR_CH_A -3

Error reading from channel A

2.4.2.11 #define READ_ERR_CH_B -4

Error reading from channel B

2.4.2.12 #define W_HIGHBYTE 0x82

Write command for highbyte (AC, BC)

2.4.2.13 #define W_LOWBYTE 0x80

Write command for lowbyte (AD, BD)

2.4.2.14 #define WRITE_ERR_CH_A -1

Error writing to channel A

2.4.2.15 #define WRITE_ERR_CH_B -2

Error writing to channel B

2.4.3 Function Documentation

2.4.3.1 void process_pins (unsigned long *ulIOMask,* unsigned long *ulOutput*)

stores a ftdi write command in a global buffer for later sending.

This function gets called repeatedly by i2c functions. It stores the commands in global Buffers (aucBufferA and aucBufferB). The commands consist of a mask which determines which pins are configured as output and input plus the actual output value to be written to the pins. All stored commands can be sent by the send_package_xx functions which form the software i2c protocol.

Parameters

in	*ulIOMask*	input / output mask to set pin functionality
in	*ulOutput*	value to be assigned to pins set as output

2.4.3.2 void process_pins_databack (unsigned long *ulIOMask,* unsigned long *ulOutput*)

stores a ftdi write command in a global buffer for later sending.

This function gets called repeatedly by i2c functions. It stores the commands in global Buffers (aucBufferA and aucBufferB). The commands consist of a mask which determines which pins are set as input and output and and output value which will be written to the pins set as output. All stored commands can be sent by the send_↩ package_xx functions which form the software i2c protocol.

Parameters

in	*ulIOMask*	input / output mask to set pin direction
in	*ulOutput*	value to be assigned to pins set as output

2.4.3.3 int readInputs (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* const unsigned char * *readBack*)

reads the input pins of both ftdi channels.

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*readBack*	contains the bytes read back from the input pins

Returns

>0 : number of bytes written to the chip
<0 : USB functions failed

2.4.3.4 int send_package_read16 (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausReadBuffer,* unsigned char *ucReadBufferLength*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip. Furthermore it reads back the data of pins which were configured as input. The function returns a value which equals the amount of read back bytes. The parameter ausReadbuffer will be used for storing 16 read back unsigned short int values of 16 sensors

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*ausReadBuffer*	pointer to array of unsigned short values
in	*ucReadBufferLength*	number of elements to be stored in ausReadbuffer

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.4.3.5 int send_package_read72 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucReadBuffer*, unsigned short * *ausReadBuffer1*, unsigned short * *ausReadBuffer2*, unsigned short * *ausReadBuffer3*, unsigned short * *ausReadBuffer4*, unsigned char *ucReadBufferLength*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip. Furthermore it reads back the data of pins which were configured as input. The function returns a value which equals the amount of read back bytes. The parameter ausReadbuffer will be used for storing 16 read back unsigned short int values of 16 sensors

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*aucReadBuffer*	pointer to array of unsigned char values
in,out	*ausReadBuffer1*	pointer to array of unsigned short values
in,out	*ausReadBuffer2*	pointer to array of unsigned short values
in,out	*ausReadBuffer3*	pointer to array of unsigned short values
in,out	*ausReadBuffer4*	pointer to array of unsigned short values
in	*ucReadBufferLength*	number of elements to be stored in ausReadbuffer

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.4.3.6 int send_package_read8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, unsigned char * *aucReadBuffer*, unsigned char *ucReadBufferLength*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip. Furthermore it reads back the data of pins which were configured as input. The function returns a value which equals the amount of read back bytes. The parameter aucReadbuffer will be used for storing 16 read back unsigned char values

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in,out	*aucReadBuffer*	pointer to array of unsigned char values
in	*ucReadBufferLength*	number of elements to be stored in aucReadbuffer

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.4.3.7 int send_package_write8 (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*)

sends the content of the global buffers to the ftdi chip.

This function sends the content of the global Buffers aucBufferA and aucBufferB to the ftdi chip Furthermore it reads back the data of pins which were configured as input. In case of i2c these read back pins can be acknowledge bits or data send back by the device.

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context

Returns

0 if succesful, errorcode if not

- WRITE_ERR_CH_A
- WRITE_ERR_CH_B
- READ_ERR_CH_A
- READ_ERR_CH_B
- ERR_INCORRECT_AMOUNT

2.4.3.8 int writeOutputs (struct ftdi_context * *ftdiA*, struct ftdi_context * *ftdiB*, const unsigned long *ulOutput*)

writes a value to the ftdi 2232h output pins.

Parameters

in	*ftdiA,ftdiB*	pointer to a ftdi_context
in	*ulOutput*	a 32 Bit value to be written to the ftdi pins
in	*ulOutput*	Bit0 will be assigned to AD0, Bit31 to BC7

Returns

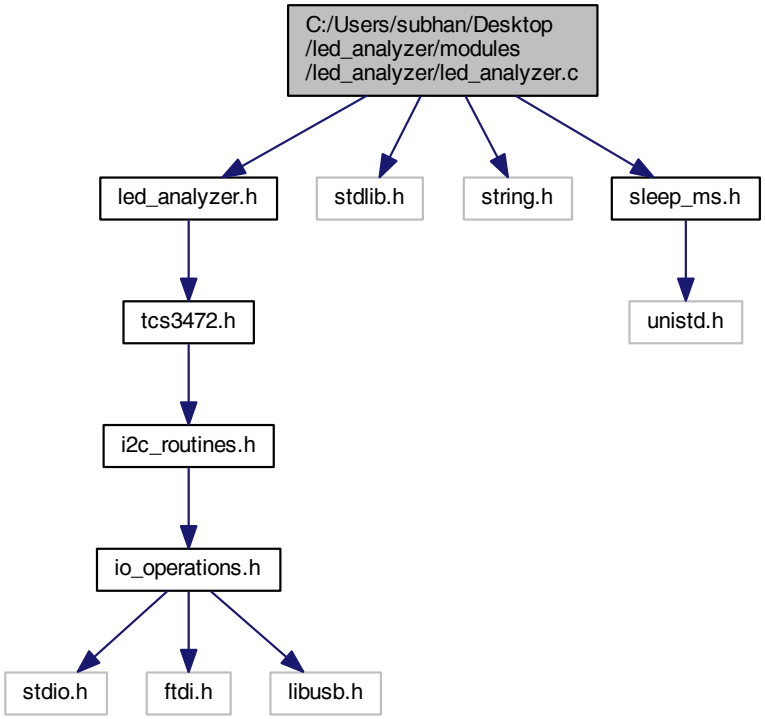
>0 : number of bytes written to the chip
<0 : USB functions failed

2.5 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/led_analyzer.c File Reference

led_analyzer handles all functionality on device level and provides the functions needed by the GUI application.

```
#include "led_analyzer.h"
#include <stdlib.h>
#include <string.h>
#include "sleep_ms.h"
```

Include dependency graph for led_analyzer.c:



Functions

- int scan_devices (char **asSerial, unsigned int asLength)

  scans for connected color controller devices and stores their serial numbers in an array.
- int connect_to_devices (void **apHandles, int apHlength, char **asSerial)

  connects to all USB devices with a given serial number.
- int get_number_of_serials (char **asSerial)

  returns number of serial numbers stored in the serial number array.
- int get_number_of_handles (void **apHandles)

  returns number of handles stored in the handle array.
- int handleToDevice (int handleIndex)

  returns the device number corresponding to a certain handleIndex.
- int init_sensors (void **apHandles, int devIndex)

  initializes the sensors of a color controller device.

- int read_colors (void **apHandles, int devIndex, unsigned short *ausClear, unsigned short *ausRed, unsigned short *ausGreen, unsigned short *ausBlue, unsigned char *aucIntegrationtime, unsigned char *aucGain)

  reads the RGBC colors of all sensors under a device and checks if the colors are valid
- void free_devices (void **apHandles)

  frees the memory of all connected opened color controller devices.
- int set_intTime_x (void **apHandles, int devIndex, unsigned int uiX, unsigned char integrationtime)

  sets the integration time of one sensor.
- int set_gain_x (void **apHandles, int devIndex, unsigned int uiX, unsigned char gain)

  sets the gain of one sensor.
- int set_gain (void **apHandles, int devIndex, unsigned char gain)

  sets the gain of 16 sensors under a device.
- int get_gain (void **apHandles, int devIndex, unsigned char *aucGains)

  reads the current gain setting of 16 sensors under a device and stores them in an adequate buffer.
- int set_intTime (void **apHandles, int devIndex, unsigned char integrationtime)

  sets the integration time of 16 sensors under a device.
- int get_intTime (void **apHandles, int devIndex, unsigned char *aucIntegrationtime)

  reads the integration time setting of 16 sensors under a device and stores them in an adequate buffer.
- void wait4Conversion (unsigned int uiWaitTime)
- int swap_serialPos (char **asSerial, unsigned int pos1, unsigned int pos2)
- int getSerialIndex (char **asSerial, char *curSerial)

  returns the index of the serial number described by curSerial.
- int swap_up (char **asSerial, char *curSerial)

  swaps the serial number described by curSerial up by one position.
- int swap_down (char **asSerial, char *curSerial)

  swaps the serial number described by curSerial down by one position.

2.5.1 Detailed Description

led_analyzer handles all functionality on device level and provides the functions needed by the GUI application.

The led_analyzer library will handle functionality to work with severeal color controller devices. It scans for connected color controller devices and opens them. Handles will be assigned to all opened color controller devices. These handles will be stored in an array and can be used by all underlying color related functions. Furthermore this library provides the functions which are required for the application CoCo App.

2.5.2 Function Documentation

2.5.2.1 int connect_to_devices (void ** *apHandles*, int *apHlength*, char ** *asSerial*)

connects to all USB devices with a given serial number.

Function opens all USB devices which have a serial number that equals one of the serial numbers given in asSerial. Furthermore it initializes the devices by configuring the right channels with the right modes. After having successfully opened a device, the handle of the opened device will be stored in apHandles. As each (ftdi2232h) color controller device has 2 channels each connected color controller will get 2 handles in apHandles.

Parameters

apHandles	stores the handles of all opened USB color controller devices

apHlength	maximum number of handles apHandles can store
asSerial	stores the serial numbers of all connected color controller devices

Return values

0	opened no color controller device
-1	error with ftdi functions or insufficient length of apHandles
>0	number of opened color controller devices

2.5.2.2 void free_devices (void ** *apHandles*)

frees the memory of all connected opened color controller devices.

Function iterates over all handle elements in apHandles and frees the memory. Freeing includes closing the usb_device and freeing the memory allocated by the device handle.

Parameters

apHandles	array that stores ftdi2232h handles

2.5.2.3 int get_gain (void ** *apHandles*, int *devIndex*, unsigned char * *aucGains*)

reads the current gain setting of 16 sensors under a device and stores them in an adequate buffer.

The function reads back the gain settings of 16 sensors. Refer to sensors' datasheet for further information about gain settings.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
aucGains	buffer which will contain the gain settings of the 16 sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.4 int get_intTime (void ** *apHandles*, int *devIndex*, unsigned char * *aucIntegrationtime*)

reads the integration time setting of 16 sensors under a device and stores them in an adequate buffer.

The function reads back the integration time of 16 sensors. Refer to sensors' datasheet for further information about integration time settings.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
aucIntegrationtime	pointer to buffer which will store the integration time settings of the 16 sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.5 int get_number_of_handles (void ** *apHandles*)

returns number of handles stored in the handle array.

Parameters

apHandles	array that stores the handles

Returns

number of elements in the handle array

2.5.2.6 int get_number_of_serials (char ** *asSerial*)

returns number of serial numbers stored in the serial number array.

Parameters

asSerial	stores the serial numbers of all connected color controller devices

Returns

number of elements in the serial number array

2.5.2.7 int getSerialIndex (char ** *asSerial,* char * *curSerial*)

returns the index of the serial number described by curSerial.

Function returns the serial number that curSerial currently has in the serial number array asSerial.

Parameters

asSerial	array which contains serial numbers of color controller devices
curSerial	current serial number

Return values

index	/ position of current serial number in asSerial

2.5.2.8 int handleToDevice (int *handleIndex*)

returns the device number corresponding to a certain handleIndex.

As each device has 2 handles, the device index and the handle index are not the same. Following functions provides an easy way to get the device number if a handle index is given

Parameters

handleIndex	index of the handle

Returns

device index that corresponds to the handle index

2.5.2.9 int init_sensors (void ** *apHandles,* int *devIndex*)

initializes the sensors of a color controller device.

Function initializes the 16 sensors of a color controller device. Initializing includes turning the sensors on clearing their interrupt flags and identifying them to be sure that the i2c-protocol works and following color readings are valid.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device

Return values

0	Succesful
>0	Flag in DWORD HIGH marks what kind of error occured, 16 bits in DWORD LOW mark which of the 16 sensors failed
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.10 int read_colors (void ** *apHandles,* int *devIndex,* unsigned short * *ausClear,* unsigned short * *ausRed,* unsigned short * *ausGreen,* unsigned short * *ausBlue,* unsigned char * *aucIntegrationtime,* unsigned char * *aucGain*)

reads the RGBC colors of all sensors under a device and checks if the colors are valid

Function reads the colors red, green, blue and clear of all 16 sensors under a device and stores them in adequate buffers. Furthermore the function will check if maximum clear levels have been exceeded. If so, the color reading won't be valid, as the sensor has been saturated, and color reading for failing sensor(s) should be redone. If maximum clear levels are beeing exceeded too often, one should consider lowering gain and/or integration time settings. The function will return a returncode which can be used to determine which of the color sensors have exceeded maximum clear levels. Furthermore the function will store the sensors' measured LUX level in an array. This level is calculated by a formula given in AMS / TAOS Designer's Note 40.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
ausClear	stores 16 clear colors
ausRed	stores 16 red colors
ausGreen	stores 16 green colors
ausBlue	stores 16 blue colors
auc↩ Integrationtime	stores 16 integration time values of the sensors
aucGain	stores 16 gain values of the sensors

Return values

0	Succesful
>0	Flag in DWORD HIGH marks what kind of error occured, 16 bits in DWORD LOW mark which of the 16 sensors failed
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.11 int scan_devices (char ** *asSerial,* unsigned int *asLength*)

scans for connected color controller devices and stores their serial numbers in an array.

Functions scans for all color controller devices with a given VID and PID that are connected via USB. A device which has "COLOR-CTRL" as description will be counted as a color controller. Function prints manufacturer, description

and serialnumber of connected devices. Furthermore the serialnumber(s) will be stored in an array and can be used by functions that open a connected device by a serialnumber.

Parameters

asSerial	stores the serial numbers of all connected color controller devices
asLength	maximum number of elements the serial number array can contain

Return values

0	no color controller device detected
<0	error with ftdi functions or insufficient space for storing all serial numbers in as↩ Serial
>0	number of connected color controller devices with given VID, PID and "COLOR-CTRL" as description

2.5.2.12 int set_gain (void ∗∗ *apHandles,* int *devIndex,* unsigned char *gain*)

sets the gain of 16 sensors under a device.

Function sets the gain of 16 sensors of a device. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
gain	gain to be sent to the sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.13 int set_gain_x (void ∗∗ *apHandles,* int *devIndex,* unsigned int *uiX,* unsigned char *gain*)

sets the gain of one sensor.

Function sets the gain of 1 sensor. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
uiX	sensor which will get the new gain (0 ... 15)
gain	gain to be sent to the sensor

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.14 int set_intTime (void ** *apHandles,* int *devIndex,* unsigned char *integrationtime*)

sets the integration time of 16 sensors under a device.

Function sets the integration time of 16 sensors. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in enum tcs3472Integration_t.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
integrationtime	integration time to be sent to the sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.15 int set_intTime_x (void ** *apHandles,* int *devIndex,* unsigned int *uiX,* unsigned char *integrationtime*)

sets the integration time of one sensor.

Function sets the integration time of one sensor. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in tcs3472Integration_t.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
uiX	sensor which will get the new integration time (0 ... 15)
integrationtime	integration time to be sent to the sensor

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.5.2.16 int swap_down (char ** *asSerial,* char * *curSerial*)

swaps the serial number described by curSerial down by one position.

Function swaps the serial number described by curSerial down by one position. The serial number that got pushed away will be in [oldPosition + 1].

Return values

0	OK - swap successful or no need to swap
-1	swapping failed

2.5.2.17 int swap_serialPos (char ** *asSerial,* unsigned int *pos1,* unsigned int *pos2*)

swaps the position of two serial numbers located in an array of serial numbers.

Function swaps the location of two serial numbers which are located in an array of serial numbers. This function will be used for having control over the opening order of usb devices, as the current algorithm iterates over the serial number array and opens the devices with those serial numbers. Thus a color controller device with a serial number at location 0 will be opened before a device with a serial number located at index 1. The handles which correspond to the opened color controller devices will be stored in the same order, as the order of opening. If the function has finished successfully the serial number which was in pos1 will be in position 2 and

Parameters

asSerial	array which contains serial numbers of color controller devices
pos1	current position of one swap operand
pos2	target position of the swap operand

Return values

0	OK - swapping successful

Returns

-1 reaching out of the serial number array

2.5.2.18 int swap_up (char ** *asSerial,* char * *curSerial*)

swaps the serial number described by curSerial up by one position.

Function swaps the serial number described by curSerial up by one position. The serial number that got pushed away will be in [oldPosition - 1].

Return values

0	OK - swap successful or no need to swap
-1	swapping failed

2.5.2.19 void wait4Conversion (unsigned int *uiWaitTime*)

waits for a time specified in uiWaitTime]1 ms ... 10 s] max

Parameters

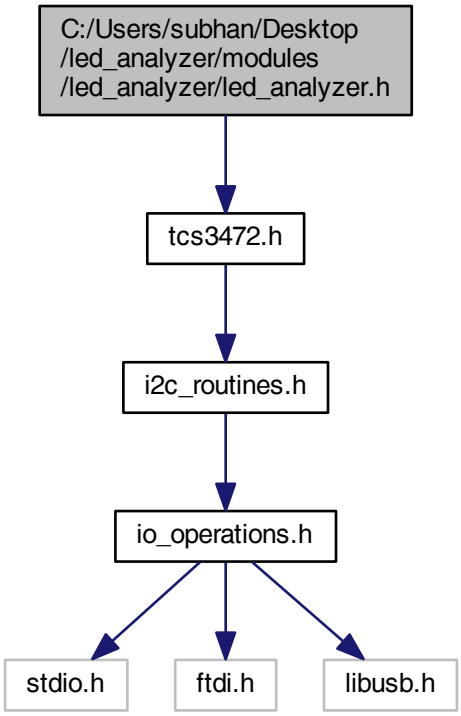
uiWaitTime	time in milliseconds to wait in order to let the sensors complete their ADC measurements

2.6 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/led_analyzer.h File Reference

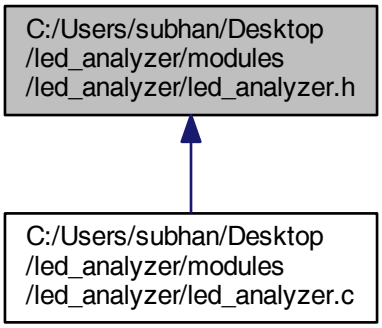
led_analyzer handles all functionality on device level and provides the functions needed by the GUI application (header)

```
#include "tcs3472.h"
```

Include dependency graph for led_analyzer.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VID 0x1939

- #define PID 0x0024
- #define MAX_DESCLENGTH 128

Enumerations

- enum E_ERROR {
  ERR_FLAG_ID = 0x40000000, ERR_FLAG_INCOMPL_CONV = 0x20000000, ERR_FLAG_EXCEEDED↩
  _CLEAR = 0x10000000, ERR_DEVICE_FATAL = 0x8000000,
  ERR_USB = 0x4000000, ERR_INDEXING = -100 }

  Contains Errorcodes and Errorflags which indicate what kind of errors occured.

Functions

- int scan_devices (char **asSerial, unsigned int uiLength)

  scans for connected color controller devices and stores their serial numbers in an array.
- int connect_to_devices (void **apHandles, int apHlength, char **asLength)

  connects to all USB devices with a given serial number.
- int read_colors (void **apHandles, int devIndex, unsigned short *ausClear, unsigned short *ausRed, unsigned short *ausGreen, unsigned short *ausBlue, unsigned char *aucIntegrationtime, unsigned char *auc↩
  Gain)

  reads the RGBC colors of all sensors under a device and checks if the colors are valid
- int init_sensors (void **apHandles, int devIndex)

  initializes the sensors of a color controller device.
- int get_number_of_handles (void **apHandles)

  returns number of handles stored in the handle array.
- int handleToDevice (int handle)

  returns the device number corresponding to a certain handleIndex.
- int set_gain (void **apHandles, int devIndex, unsigned char gain)

  sets the gain of 16 sensors under a device.
- int set_gain_x (void **apHandles, int devIndex, unsigned int uiX, unsigned char gain)

  sets the gain of one sensor.
- int get_gain (void **apHandles, int devIndex, unsigned char *aucGains)

  reads the current gain setting of 16 sensors under a device and stores them in an adequate buffer.
- int set_intTime (void **apHandles, int devIndex, unsigned char integrationtime)

  sets the integration time of 16 sensors under a device.
- int set_intTime_x (void **apHandles, int devIndex, unsigned int uiX, unsigned char integrationtime)

  sets the integration time of one sensor.
- int get_intTime (void **apHandles, int devIndex, unsigned char *aucIntTimeSettings)

  reads the integration time setting of 16 sensors under a device and stores them in an adequate buffer.
- int get_number_of_serials (char **asSerial)

  returns number of serial numbers stored in the serial number array.
- int swap_serialPos (char **asSerial, unsigned int swap1, unsigned int swap2)
- int getSerialIndex (char **asSerial, char *curSerial)

  returns the index of the serial number described by curSerial.
- int swap_up (char **asSerial, char *curSerial)

  swaps the serial number described by curSerial up by one position.
- int swap_down (char **asSerial, char *curSerial)

  swaps the serial number described by curSerial down by one position.
- void free_devices (void **apHandles)

  frees the memory of all connected opened color controller devices.
- void wait4Conversion (unsigned int uiWaitTime)

2.6.1 Detailed Description

led_analyzer handles all functionality on device level and provides the functions needed by the GUI application (header)

The led_analyzer library will handle functionality to work with severeal color controller devices. It scans for connected color controller devices and opens them. All devices that have been connected to will have handles. These handles will be stored in an array and can be used by all underlying color related functions. Furthermore this library provides the functions which are required for the application CoCo App.

2.6.2 Macro Definition Documentation

2.6.2.1 #define MAX_DESCLENGTH 128

Maximum Length of characters a descriptor in the ftdi2232h eeprom can have

2.6.2.2 #define PID 0x0024

Product ID for the Color Controller "COLOR-CTRL"

2.6.2.3 #define VID 0x1939

Vendor ID of 'Hilscher Gesellschaft für Systemautomation mbH'

2.6.3 Enumeration Type Documentation

2.6.3.1 enum E_ERROR

Contains Errorcodes and Errorflags which indicate what kind of errors occured.

The errorflags indicate what kind of error occured. They get ored with the erroflag of the sensors in order to show what kind of error occured with what sensor. Furthermore there are errorcodes which indicate errors on usb, i2c and device level or if indexing outside array boundaries occur.

Enumerator

ERR_FLAG_ID Flag - Identification error occured, e.g. the ID register value couldn't be read

ERR_FLAG_INCOMPL_CONV Flag - The conversion was not complete at the time the ADC register was accessed

ERR_FLAG_EXCEEDED_CLEAR The maximum amount of clear level was reached, i.e. the sensor got digitally saturated

ERR_DEVICE_FATAL Errorcode - Fatal error on a device, writing / reading from a ftdi channel failed

ERR_USB Errorcode - USB error on a device, which means that we read back a different number of bytes than we expected to read

ERR_INDEXING Indexing outside the handles array (apHandles)

2.6.4 Function Documentation

2.6.4.1 int connect_to_devices (void ** *apHandles,* int *apHlength,* char ** *asSerial*)

connects to all USB devices with a given serial number.

Function opens all USB devices which have a serial number that equals one of the serial numbers given in as↩ Serial. Furthermore it initializes the devices by configuring the right channels with the right modes. After having

successfully opened a device, the handle of the opened device will be stored in apHandles. As each (ftdi2232h) color controller device has 2 channels each connected color controller will get 2 handles in apHandles.

Parameters

apHandles	stores the handles of all opened USB color controller devices
apHlength	maximum number of handles apHandles can store
asSerial	stores the serial numbers of all connected color controller devices

Return values

0	opened no color controller device
-1	error with ftdi functions or insufficient length of apHandles
>0	number of opened color controller devices

2.6.4.2 void free_devices (void ** *apHandles*)

frees the memory of all connected opened color controller devices.

Function iterates over all handle elements in apHandles and frees the memory. Freeing includes closing the usb↩ _device and freeing the memory allocated by the device handle.

Parameters

apHandles	array that stores ftdi2232h handles

2.6.4.3 int get_gain (void ** *apHandles,* int *devIndex,* unsigned char * *aucGains*)

reads the current gain setting of 16 sensors under a device and stores them in an adequate buffer.

The function reads back the gain settings of 16 sensors. Refer to sensors' datasheet for further information about gain settings.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
aucGains	buffer which will contain the gain settings of the 16 sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.4 int get_intTime (void ** *apHandles,* int *devIndex,* unsigned char * *aucIntegrationtime*)

reads the integration time setting of 16 sensors under a device and stores them in an adequate buffer.

The function reads back the integration time of 16 sensors. Refer to sensors' datasheet for further information about integration time settings.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
auc↩ Integrationtime	pointer to buffer which will store the integration time settings of the 16 sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.5 int get_number_of_handles (void ** *apHandles*)

returns number of handles stored in the handle array.

Parameters

apHandles	array that stores the handles

Returns

number of elements in the handle array

2.6.4.6 int get_number_of_serials (char ** *asSerial*)

returns number of serial numbers stored in the serial number array.

Parameters

asSerial	stores the serial numbers of all connected color controller devices

Returns

number of elements in the serial number array

2.6.4.7 int getSerialIndex (char ** *asSerial,* char * *curSerial*)

returns the index of the serial number described by curSerial.

Function returns the serial number that curSerial currently has in the serial number array asSerial.

Parameters

asSerial	array which contains serial numbers of color controller devices
curSerial	current serial number

Return values

index	/ position of current serial number in asSerial

2.6.4.8 int handleToDevice (int *handleIndex*)

returns the device number corresponding to a certain handleIndex.

As each device has 2 handles, the device index and the handle index are not the same. Following functions provides an easy way to get the device number if a handle index is given

Parameters

handleIndex	index of the handle

Returns

device index that corresponds to the handle index

2.6.4.9 int init_sensors (void ∗∗ *apHandles,* int *devIndex*)

initializes the sensors of a color controller device.

Function initializes the 16 sensors of a color controller device. Initializing includes turning the sensors on clearing their interrupt flags and identifying them to be sure that the i2c-protocol works and following color readings are valid.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device

Return values

0	Succesful
>0	Flag in DWORD HIGH marks what kind of error occured, 16 bits in DWORD LOW mark which of the 16 sensors failed
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.10 int read_colors (void ∗∗ *apHandles,* int *devIndex,* unsigned short ∗ *ausClear,* unsigned short ∗ *ausRed,* unsigned short ∗ *ausGreen,* unsigned short ∗ *ausBlue,* unsigned char ∗ *aucIntegrationtime,* unsigned char ∗ *aucGain*)

reads the RGBC colors of all sensors under a device and checks if the colors are valid

Function reads the colors red, green, blue and clear of all 16 sensors under a device and stores them in adequate buffers. Furthermore the function will check if maximum clear levels have been exceeded. If so, the color reading won't be valid, as the sensor has been saturated, and color reading for failing sensor(s) should be redone. If maximum clear levels are beeing exceeded too often, one should consider lowering gain and/or integration time settings. The function will return a returncode which can be used to determine which of the color sensors have exceeded maximum clear levels. Furthermore the function will store the sensors' measured LUX level in an array. This level is calculated by a formula given in AMS / TAOS Designer's Note 40.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
ausClear	stores 16 clear colors
ausRed	stores 16 red colors
ausGreen	stores 16 green colors
ausBlue	stores 16 blue colors
auc↩ Integrationtime	stores 16 integration time values of the sensors
aucGain	stores 16 gain values of the sensors

Return values

0	Succesful

>0	Flag in DWORD HIGH marks what kind of error occured, 16 bits in DWORD LOW mark which of the 16 sensors failed
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.11 int scan_devices (char ** *asSerial,* unsigned int *asLength*)

scans for connected color controller devices and stores their serial numbers in an array.

Functions scans for all color controller devices with a given VID and PID that are connected via USB. A device which has "COLOR-CTRL" as description will be counted as a color controller. Function prints manufacturer, description and serialnumber of connected devices. Furthermore the serialnumber(s) will be stored in an array and can be used by functions that open a connected device by a serialnumber.

Parameters

asSerial	stores the serial numbers of all connected color controller devices
asLength	maximum number of elements the serial number array can contain

Return values

0	no color controller device detected
<0	error with ftdi functions or insufficient space for storing all serial numbers in asSerial
>0	number of connected color controller devices with given VID, PID and "COLOR-CTRL" as description

2.6.4.12 int set_gain (void ** *apHandles,* int *devIndex,* unsigned char *gain*)

sets the gain of 16 sensors under a device.

Function sets the gain of 16 sensors of a device. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
gain	gain to be sent to the sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.13 int set_gain_x (void ** *apHandles,* int *devIndex,* unsigned int *uiX,* unsigned char *gain*)

sets the gain of one sensor.

Function sets the gain of 1 sensor. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
uiX	sensor which will get the new gain (0 ... 15)
gain	gain to be sent to the sensor

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.14 int set_intTime (void ** *apHandles,* int *devIndex,* unsigned char *integrationtime*)

sets the integration time of 16 sensors under a device.

Function sets the integration time of 16 sensors. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in enum tcs3472Integration_t.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
integrationtime	integration time to be sent to the sensors

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.15 int set_intTime_x (void ** *apHandles,* int *devIndex,* unsigned int *uiX,* unsigned char *integrationtime*)

sets the integration time of one sensor.

Function sets the integration time of one sensor. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in tcs3472Integration_t.

Parameters

apHandles	array that stores ftdi2232h handles
devIndex	device index of current color controller device
uiX	sensor which will get the new integration time (0 ... 15)
integrationtime	integration time to be sent to the sensor

Return values

0	Succesful
<0	USB errors, i2c errors or indexing errors occured, check return value for further information

2.6.4.16 int swap_down (char ** *asSerial*, char * *curSerial*)

swaps the serial number described by curSerial down by one position.

Function swaps the serial number described by curSerial down by one position. The serial number that got pushed away will be in [oldPosition + 1].

Return values

0	OK - swap successful or no need to swap
-1	swapping failed

2.6.4.17 int swap_serialPos (char ** *asSerial*, unsigned int *pos1*, unsigned int *pos2*)

swaps the position of two serial numbers located in an array of serial numbers.

Function swaps the location of two serial numbers which are located in an array of serial numbers. This function will be used for having control over the opening order of usb devices, as the current algorithm iterates over the serial number array and opens the devices with those serial numbers. Thus a color controller device with a serial number at location 0 will be opened before a device with a serial number located at index 1. The handles which correspond to the opened color controller devices will be stored in the same order, as the order of opening. If the function has finished successfully the serial number which was in pos1 will be in position 2 and

Parameters

asSerial	array which contains serial numbers of color controller devices
pos1	current position of one swap operand
pos2	target position of the swap operand

Return values

0	OK - swapping successful

Returns

-1 reaching out of the serial number array

2.6.4.18 int swap_up (char ** *asSerial*, char * *curSerial*)

swaps the serial number described by curSerial up by one position.

Function swaps the serial number described by curSerial up by one position. The serial number that got pushed away will be in [oldPosition - 1].

Return values

0	OK - swap successful or no need to swap
-1	swapping failed

2.6.4.19 void wait4Conversion (unsigned int *uiWaitTime*)

waits for a time specified in uiWaitTime]1 ms ... 10 s] max

Parameters

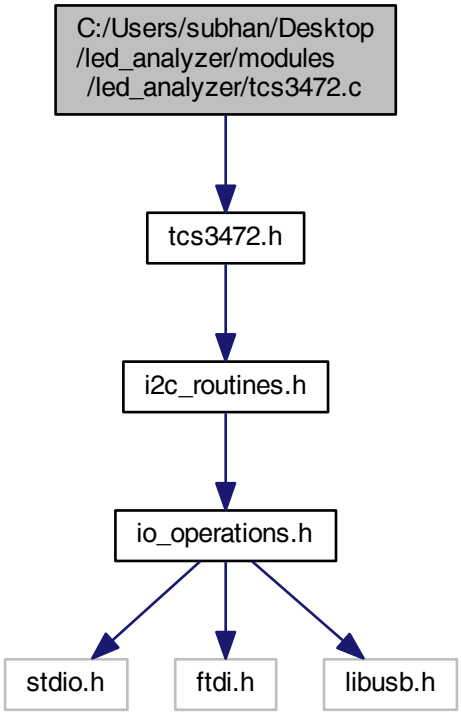
uiWaitTime	time in milliseconds to wait in order to let the sensors complete their ADC measurements

2.7 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/tcs3472.c File Reference

Library to operate with 16 AMS/TAOS Color Sensors (TCS3472)

```
#include "tcs3472.h"
```

Include dependency graph for tcs3472.c:



Functions

- int tcs_identify (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucReadbuffer)

  reads the ID-Register of 16 sensors and compares the values to expected values.
- int tcs_ON (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)

  turns 16 tcs3472 sensors on, releasing them from their sleep state.
- int tcs_setIntegrationTime (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Integration_t uiIntegrationtime)

  sets the integration time of 16 sensors at once.
- int tcs_setIntegrationTime_x (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Integration_t uiIntegrationtime, unsigned int uiX)

  sets the integration time of one sensor.

- int tcs_setGain (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Gain_t gain)
  sets the gain of 16 sensors.
- int tcs_setGain_x (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Gain_t gain, unsigned int uiX)
  sets the gain of one sensor.
- int tcs_waitForData (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  checks if the ADCs for color measurement have already completed.
- int tcs_conversions_complete (unsigned char *aucStatusRegister)
  checks if the ADCs for color measurement have already completed. Takes the status register as parameter.
- int tcs_readColor (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausColorArray, tcs_↩ color_t color)
  reads back 4 color sets of 16 sensors - Red / Green / Blue / Clear.
- int tcs_readColors (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausClear, unsigned short *ausRed, unsigned short *ausGreen, unsigned short *ausBlue)
  reads back 4 colours and the content of the status register in one i2c command.
- int tcs_sleep (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  sends 16 sensors to sleep.
- int tcs_wakeUp (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  wakes up 16 color sensors.
- int tcs_exClear (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausClear, unsigned char *aucIntegrationtime)
  checks the clear color read back from the sensors have exceeded maximum clear.
- void tcs_calculate_CCT_Lux (unsigned char *aucGain, unsigned char *aucIntegrationtime, unsigned short *ausClear, unsigned short *ausRed, unsigned short *ausGreen, unsigned short *ausBlue, unsigned short *CCT, float *afLUX)
  calculates Illuminance in unit LUX.
- int tcs_clearInt (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  clears the interrupt flag of 16 sensors.
- int tcs_getGain (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucGainSettings)
  reads the current gain setting of 16 sensors and stores them in an adequate buffer.
- int tcs_getIntegrationtime (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *auc↩ Integrationtime)
  reads the current integration time setting of 16 sensors and stores them in an adequate buffer.
- int getGainDivisor (tcs3472Gain_t gain)
  returns a divisor which corresponds to a specific gain setting.

2.7.1 Detailed Description

Library to operate with 16 AMS/TAOS Color Sensors (TCS3472)

tcs3472 provides functionality to address the color sensor tcs3472, an RGBC sensor manufactured by TAOS. Functions include identifying, turning on/off, setting integration time and gain, reading colors and checking colors for their validity. The library intends to address 16 color sensors at once. In case any of the functions fail, an return code will be returned which can be used to determine which specific sensor(s) failed. Most of the functions have 2 pointers to ftdi contexts as parameters. Each pointer represents a channel of the ftdi 2232h chip, thus controls 8 sensors.

2.7.2 Function Documentation

2.7.2.1 int getGainDivisor (tcs3472Gain_t *gain*)

returns a divisor which corresponds to a specific gain setting.

As the actual gain value which will be written into the sensors' register does not match its enum name (i.e. TC↩ S3472_GAIN_1X has the value 0) this function is needed to determine the divisor that corresponds to a specific gain setting (i.e. returns 1 for TCS3472_GAIN_1X)

Parameters

gain	enum value for gain setting

Returns

gain divisor value which corresponds to the enum value of the gain setting

2.7.2.2 void tcs_calculate_CCT_Lux (unsigned char * *aucGain,* unsigned char * *aucIntegrationtime,* unsigned short * *ausClear,* unsigned short * *ausRed,* unsigned short * *ausGreen,* unsigned short * *ausBlue,* unsigned short * *CCT,* float * *afLUX*)

calculates Illuminance in unit LUX.

Functions calculates the illuminance level of 16 sensors. In photometry, illuminance is the total luminous flux incident on a surface, per unit area. It is a measure of how much the incident light illuminates the surface, wavelength-weighted by the luminosity function to correlate with human brightness perception. Color temperature is related to the color with which a piece of metal glows when heated to a particular temperature and is typically stated in terms of degrees Kelvin. The color temperature goes from red at lower temperatures to blue at higher temperatures. Refer to Design Note 40 from AMS / Taos for further information about the calculations.

Parameters

aucGain	current gain setting of 16 sensors
auc↩ Integrationtime	current integration time setting of 16 sensors
ausClear	contains clear value of 16 sensors
ausRed	contains red value of 16 sensors
ausGreen	contains green value of 16 sensors
ausBlue	contains blue value of 16 sensors
afLUX	will store the calculated LUX values of 16 sensors
CCT	will store the calculated CCT values of 16 sensors

2.7.2.3 int tcs_clearInt (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

clears the interrupt flag of 16 sensors.

Functions clears the interrupt flag of 16 sensors. The flag will be set if a color value has been exceeded, or the value read back from the sensor has fallen below a certain color value. Both settings can be set up in the sensors' registers.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Returns

0 : everything OK - conversions complete
1 : i2c-functions failed

2.7.2.4 int tcs_conversions_complete (unsigned char * *aucStatusRegister*)

checks if the ADCs for color measurement have already completed. Takes the status register as parameter.

Function checks if the sensors have already completed a color measuremend. In case the measurements are completed the ADCs can be safely read and used for further calculations. This can be checked by reading a special register of the sensor, the status register. If TCS3472_AVALID_BIT is set in this register, the ADCs have completed color measurements. If measurements are not completed, the return code can be used to determine which of the 16 sensor(s) failed.

Parameters

aucStatus↩ Register	holds the values of the status register for all 16 sensors

Return values

0	Succesful
>0	One or more sensors have not completed the conversion cycle yet, if the return code is 0b0000000000101100 for example, we have uncompleted conversions for sensor 3, sensor 4 and sensor 6

2.7.2.5 int tcs_exClear (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausClear,* unsigned char * *aucIntegrationtime*)

checks the clear color read back from the sensors have exceeded maximum clear.

Functions checks if the clear color read back from 16 sensors have exceeded a maximum clear level and are thus invalid. In case this maximum clear level is exceeded, one should consider lowering gain and/or integration time. If clear levels have been exceeded, the return code can be used to determine which of the 16 sensor(s) failed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
ausClear	clear values of the 16 sensors which will be checked for maximum clear level exceedings
auc↩ Integrationtime	current integration time setting of the 16 sensors - needed to check if maximum clear level has been exceeded

Return values

0	Succesful
>0	One or more sensors got saturated as they have reached the maximum amount in the clear data register, if the return code is 0b0000000000101100 for example, sensor 3, sensor 4 and sensor 6 got saturated

2.7.2.6 int tcs_getGain (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucGainSettings*)

reads the current gain setting of 16 sensors and stores them in an adequate buffer.

The function reads back the gain settings of 16 sensors. Refer to sensors' datasheet for further information about gain settings.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucGainSettings	pointer to buffer which will contain the gain settings of the 16 sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.7 int tcs_getIntegrationtime (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucIntegrationtime*)

reads the current integration time setting of 16 sensors and stores them in an adequate buffer.

The function reads back the integration time of 16 sensors. Refer to sensors' datasheet for further information about integration time settings.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
auc↩ Integrationtime	pointer to buffer which will store the integration time settings of the 16 sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.8 int tcs_identify (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucReadbuffer*)

reads the ID-Register of 16 sensors and compares the values to expected values.

Expected ID for the TCS3472 is 0x44, the expected ID for the TCS3471 is 0x14

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucReadbuffer	stores the ID-values read back from the 16 sensors, must be able to hold 16 elements

Return values

0	Succesful
>0	Failed to identify one or more sensors, if the return code is 0b0000000000101100 for example, identification failed for sensor 3, sensor 4 and sensor 6
<0	USB or i2c errors occured, check return value for further information

2.7.2.9 int tcs_ON (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

turns 16 tcs3472 sensors on, releasing them from their sleep state.

Function wakes 16 sensors on in case they were put to sleep before. If sensors are already active this function has no effect.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.10 int tcs_readColor (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausColorArray,* tcs_color_t *color*)

reads back 4 color sets of 16 sensors - Red / Green / Blue / Clear.

Function reads 16-Bit color values of 16 sensors. The color will be specified by the input parameter tcs_color_t color.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

ausColorArray	will contain color value read back from 16 sensors
color	specifies the color to be read from the sensor (red, green, blue, clear)

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.11 int tcs_readColors (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausClear,* unsigned short * *ausRed,* unsigned short * *ausGreen,* unsigned short * *ausBlue*)

reads back 4 colours and the content of the status register in one i2c command.

This function will read out the contents of the color registers of tcs3472 as words (2 Bytes per colour) and the content of the status register as one byte. The status register can be used in order to determine if color conversions had already completed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
ausClear	will contain color value read back from 16 sensors
ausRed	will contain color value read back from 16 sensors
ausGreen	will contain color value read back from 16 sensors
ausBlue	will contain color value read back from 16 sensors

```
@retval 0  Succesful
@retval <0 USB or i2c errors occured, check return value for further information
@retval >0 One or more sensors have not completed the conversion cycle yet,
           if the return code is 0b0000000000101100 for example, we have uncompleted conversions for sensor 3,
```

2.7.2.12 int tcs_setGain (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Gain_t *gain*)

sets the gain of 16 sensors.

Function sets the gain of 16 sensors. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
gain	gain to be sent to the sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.13 int tcs_setGain_x (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Gain_t *gain,* unsigned int *uiX*)

sets the gain of one sensor.

Function sets the gain of 1 sensors This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
gain	gain to be sent to the sensor
uiX	sensor which will get the new gain (0 ... 15)

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.14 int tcs_setIntegrationTime (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Integration_t *uiIntegrationtime*)

sets the integration time of 16 sensors at once.

Function sets the integration time of 16 sensors. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in enum tcs3472Integration_t.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
uiIntegrationtime	integration time to be sent to the sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.15 int tcs_setIntegrationTime_x (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Integration_t *uiIntegrationtime,* unsigned int *uiX*)

sets the integration time of one sensor.

Function sets the integration time of one sensor. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in enum tcs3472Integration_t.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
uiIntegrationtime	integration time to be sent to the sensor
uiX	sensor which will get the new integration time (0 ... 15)

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.16 int tcs_sleep (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

sends 16 sensors to sleep.

Function sends 16 color sensors to sleep state.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.7.2.17 int tcs_waitForData (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

checks if the ADCs for color measurement have already completed.

Function checks if the sensors have already completed a color measuremend. In case the measurements are completed the ADCs can be safely read and the datasets are valid. This can be checked by reading a special register of the sensor, the status register. If TCS3472_AVALID_BIT is set in this register, the ADCs have completed color measurements. If measurements are not completed, the return code can be used to determine which of the 16 sensor(s) failed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
>0	One or more sensors have not completed the conversion cycle yet, if the return code is 0b0000000000101100 for example, we have uncompleted conversions for sensor 3, sensor 4 and sensor 6
<0	USB or i2c errors occured, check return value for further information

2.7.2.18 int tcs_wakeUp (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

wakes up 16 color sensors.

Function wakes 16 color sensors from sleep state.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

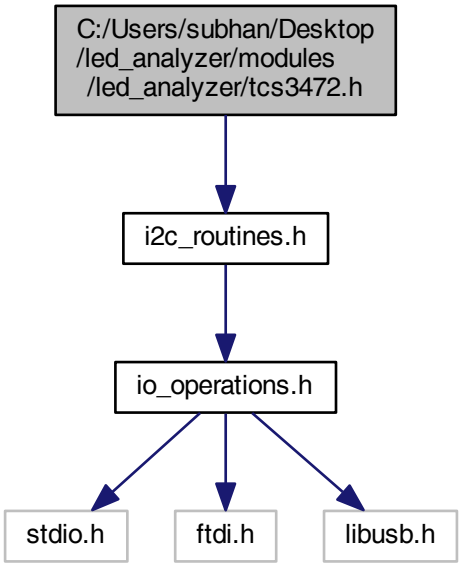
0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8 C:/Users/subhan/Desktop/led_analyzer/modules/led_analyzer/tcs3472.h File Reference

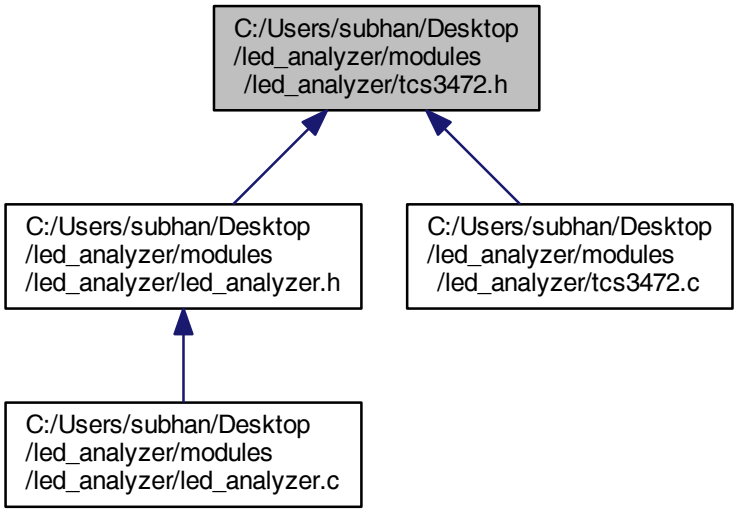
Library to operate with 16 AMS/TAOS Color Sensors (TCS3472) (header)

```
#include "i2c_routines.h"
```

Include dependency graph for tcs3472.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define TCS_ADDRESS 0x29
- #define TCS3472_ENABLE_REG 0x00
- #define TCS3472_ATIME_REG 0x01
- #define TCS3472_WTIME_REG 0x03
- #define TCS3472_AILTL_REG 0x04
- #define TCS3472_AILTH_REG 0x05
- #define TCS3472_AIHTL_REG 0x06
- #define TCS3472_AIHTH_REG 0x07
- #define TCS3472_PERS_REG 0x0C
- #define TCS3472_CONFIG_REG 0x0D
- #define TCS3472_CONTROL_REG 0x0F
- #define TCS3472_ID_REG 0x12
- #define TCS3472_STATUS_REG 0x13
- #define TCS3472_CDATA_REG 0x14
- #define TCS3472_CDATAH_REG 0x15
- #define TCS3472_RDATA_REG 0x16
- #define TCS3472_RDATAH_REG 0x17
- #define TCS3472_GDATA_REG 0x18
- #define TCS3472_GDATAH_REG 0x19
- #define TCS3472_BDATA_REG 0x1A
- #define TCS3472_BDATAH_REG 0x1B
- #define TCS3472_COMMAND_BIT 0x80
- #define TCS3472_AUTOINCR_BIT 0x20
- #define TCS3472_SPECIAL_BIT 0x60
- #define TCS3472_INTCLEAR_BIT 0x06
- #define TCS3472_AIEN_BIT 0x10
- #define TCS3472_WEN_BIT 0x08
- #define TCS3472_AEN_BIT 0x02
- #define TCS3472_PON_BIT 0x01
- #define TCS3472_WLONG_BIT 0x02
- #define TCS3472_1_5_VALUE 0x14
- #define TCS3472_3_7_VALUE 0x1D
- #define TCS3472_AINT_BIT 0x10
- #define TCS3472_AVALID_BIT 0x01

Enumerations

- enum tcs3472Gain_t {
  TCS3472_GAIN_1X = 0x00, TCS3472_GAIN_4X = 0x01, TCS3472_GAIN_16X = 0x02, TCS3472_GAIN_↩
  60X = 0x03,
  TCS3472_GAIN_ERROR = 0xFF }

  contains the gain setting commands for the sensor
- enum tcs3472Integration_t {
  TCS3472_INTEGRATION_2_4ms = 0xFF, TCS3472_INTEGRATION_24ms = 0xF6, TCS3472_INTEGRA↩
  TION_100ms = 0xD6, TCS3472_INTEGRATION_154ms = 0xC0,
  TCS3472_INTEGRATION_200ms = 0xAD, TCS3472_INTEGRATION_700ms = 0x00 }

  contains the integration time setting commands for the sensor
- enum tcs_color_t { RED = 0x00, GREEN = 0x01, BLUE = 0x02, CLEAR = 0x03 }

  commands for different color readings (Red, Green, Blue, Clear)

Functions

- int tcs_identify (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucReadbuffer)
  reads the ID-Register of 16 sensors and compares the values to expected values.
- int tcs_waitForData (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  checks if the ADCs for color measurement have already completed.
- int tcs_conversions_complete (unsigned char *aucStatusRegister)
  checks if the ADCs for color measurement have already completed. Takes the status register as parameter.
- int tcs_readColor (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausColourArray, tcs_color_t color)
  reads back 4 color sets of 16 sensors - Red / Green / Blue / Clear.
- int tcs_sleep (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  sends 16 sensors to sleep.
- int tcs_wakeUp (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  wakes up 16 color sensors.
- int tcs_ON (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  turns 16 tcs3472 sensors on, releasing them from their sleep state.
- int tcs_exClear (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausClear, unsigned char *aucIntegrationtime)
  checks the clear color read back from the sensors have exceeded maximum clear.
- int tcs_clearInt (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB)
  clears the interrupt flag of 16 sensors.
- int getGainDivisor (tcs3472Gain_t gain)
  returns a divisor which corresponds to a specific gain setting.
- int tcs_getIntegrationtime (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucIntegrationtime)
  reads the current integration time setting of 16 sensors and stores them in an adequate buffer.
- int tcs_getGain (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned char *aucGainSettings)
  reads the current gain setting of 16 sensors and stores them in an adequate buffer.
- int tcs_setIntegrationTime (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Integration_t integration)
  sets the integration time of 16 sensors at once.
- int tcs_setIntegrationTime_x (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Integration_t integration, unsigned int uiX)
  sets the integration time of one sensor.
- int tcs_setGain (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Gain_t gain)
  sets the gain of 16 sensors.
- int tcs_setGain_x (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, tcs3472Gain_t gain, unsigned int uiX)
  sets the gain of one sensor.
- int tcs_readColors (struct ftdi_context *ftdiA, struct ftdi_context *ftdiB, unsigned short *ausClear, unsigned short *ausRed, unsigned short *ausGreen, unsigned short *ausBlue)
  reads back 4 colours and the content of the status register in one i2c command.
- void tcs_calculate_CCT_Lux (unsigned char *aucGain, unsigned char *aucIntegrationtime, unsigned short *ausClear, unsigned short *ausRed, unsigned short *ausGreen, unsigned short *ausBlue, unsigned short *CCT, float *afLUX)
  calculates Illuminance in unit LUX.

2.8.1 Detailed Description

Library to operate with 16 AMS/TAOS Color Sensors (TCS3472) (header)

tcs3472 provides functionality to address the color sensor tcs3472, an RGBC sensor manufactured by TAOS. Functions include identifying, turning on/off, setting integration time and gain, reading colors and checking read colors for their validity. The library is written to address 16 color sensors at once. In case any of the functions fail, an return code will be returned which can be used to determine which specific sensor(s) failed.

2.8.2 Macro Definition Documentation

2.8.2.1 #define TCS3472_1_5_VALUE 0x14

ID register value for tcs347215

2.8.2.2 #define TCS3472_3_7_VALUE 0x1D

ID register value for tcs347237

2.8.2.3 #define TCS3472_AEN_BIT 0x02

RGBC enable bit

2.8.2.4 #define TCS3472_AIEN_BIT 0x10

RGBC interrupt enable bit

2.8.2.5 #define TCS3472_AIHTH_REG 0x07

RGBC interrupt high threshold high byte

2.8.2.6 #define TCS3472_AIHTL_REG 0x06

RGBC interrupt high threshold low byte

2.8.2.7 #define TCS3472_AILTH_REG 0x05

RGBC interrupt low threshold high byte

2.8.2.8 #define TCS3472_AILTL_REG 0x04

RGBC interrupt low threshold low byte

2.8.2.9 #define TCS3472_AINT_BIT 0x10

RGBC clear channel interrupt bit

2.8.2.10 #define TCS3472_ATIME_REG 0x01

integration time register

2.8.2.11 #define TCS3472_AUTOINCR_BIT 0x20

autoincrement bit

2.8.2.12 #define TCS3472_AVALID_BIT 0x01

RGBC valid bit - indicates that RGBC have completed an integration cycle

2.8.2.13 #define TCS3472_BDATA_REG 0x1A

blue ADC low data register

2.8.2.14 #define TCS3472_BDATAH_REG 0x1B

blue ADC high data register

2.8.2.15 #define TCS3472_CDATA_REG 0x14

clear ADC low data register

2.8.2.16 #define TCS3472_CDATAH_REG 0x15

clear ADC high data register

2.8.2.17 #define TCS3472_COMMAND_BIT 0x80

command bit

2.8.2.18 #define TCS3472_CONFIG_REG 0x0D

configuration register

2.8.2.19 #define TCS3472_CONTROL_REG 0x0F

gain control register

2.8.2.20 #define TCS3472_ENABLE_REG 0x00

Enable register

2.8.2.21 #define TCS3472_GDATA_REG 0x18

green ADC low data register

2.8.2.22 #define TCS3472_GDATAH_REG 0x19

green ADC high data register

2.8.2.23 #define TCS3472_ID_REG 0x12

device ID register

2.8.2.24 #define TCS3472_INTCLEAR_BIT 0x06

interrupt clear bit

2.8.2.25 #define TCS3472_PERS_REG 0x0C

interrupt persistence filters

2.8.2.26 #define TCS3472_PON_BIT 0x01

power on bit - activates oscillator

2.8.2.27 #define TCS3472_RDATA_REG 0x16

red ADC low data register

2.8.2.28 #define TCS3472_RDATAH_REG 0x17

red ADC high data register

2.8.2.29 #define TCS3472_SPECIAL_BIT 0x60

special bit

2.8.2.30 #define TCS3472_STATUS_REG 0x13

device status register

2.8.2.31 #define TCS3472_WEN_BIT 0x08

device wait enable bit

2.8.2.32 #define TCS3472_WLONG_BIT 0x02

wait long bit

2.8.2.33 #define TCS3472_WTIME_REG 0x03

wait time register

2.8.2.34 #define TCS_ADDRESS 0x29

i2c-address of the tcs3472 color sensor

2.8.3 Enumeration Type Documentation

2.8.3.1 enum tcs3472Gain_t

contains the gain setting commands for the sensor

Gain setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain setting.

Enumerator

TCS3472_GAIN_1X 1X Gain

TCS3472_GAIN_4X 4X Gain

TCS3472_GAIN_16X 16X Gain

TCS3472_GAIN_60X 60X Gain

TCS3472_GAIN_ERROR ERROR

2.8.3.2 enum tcs3472Integration_t

contains the integration time setting commands for the sensor

Integration time setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, integration time for bright LEDs can be low. Refer to the sensor's datasheet for further information about the integration time setting. Calculation of the register content for a specific integration time can be found in the datasheet as well. This enum contains 6 different integration time settings, though there's any setting possible starting from 2.4ms to 700ms (in 2.4ms steps).

Enumerator

TCS3472_INTEGRATION_2_4ms command for 2.4 milliseconds integration time

TCS3472_INTEGRATION_24ms command for 24 milliseconds integration time

TCS3472_INTEGRATION_100ms command for 100 milliseconds integration time

TCS3472_INTEGRATION_154ms command for 154 milliseconds integration time

TCS3472_INTEGRATION_200ms command for 200 milliseconds integration time

TCS3472_INTEGRATION_700ms command for 700 milliseconds integration time

2.8.3.3 enum tcs_color_t

commands for different color readings (Red, Green, Blue, Clear)

Enumerator

RED Command for Red Color Reading

GREEN Command for Green Color Reading

BLUE Command for Blue Color Reading

CLEAR Command for Clear Color Reading

2.8.4 Function Documentation

2.8.4.1 int getGainDivisor (tcs3472Gain_t *gain*)

returns a divisor which corresponds to a specific gain setting.

As the actual gain value which will be written into the sensors' register does not match its enum name (i.e. TC↩S3472_GAIN_1X has the value 0) this function is needed to determine the divisor that corresponds to a specific gain setting (i.e. returns 1 for TCS3472_GAIN_1X)

Parameters

gain	enum value for gain setting

Returns

gain divisor value which corresponds to the enum value of the gain setting

2.8.4.2 void tcs_calculate_CCT_Lux (unsigned char * *aucGain,* unsigned char * *aucIntegrationtime,* unsigned short * *ausClear,* unsigned short * *ausRed,* unsigned short * *ausGreen,* unsigned short * *ausBlue,* unsigned short * *CCT,* float * *afLUX*)

calculates Illuminance in unit LUX.

Functions calculates the illuminance level of 16 sensors. In photometry, illuminance is the total luminous flux incident on a surface, per unit area. It is a measure of how much the incident light illuminates the surface, wavelength-weighted by the luminosity function to correlate with human brightness perception. Color temperature is related to the color with which a piece of metal glows when heated to a particular temperature and is typically stated in terms of degrees Kelvin. The color temperature goes from red at lower temperatures to blue at higher temperatures. Refer to Design Note 40 from AMS / Taos for further information about the calculations.

Parameters

aucGain	current gain setting of 16 sensors
auc↩ Integrationtime	current integration time setting of 16 sensors
ausClear	contains clear value of 16 sensors
ausRed	contains red value of 16 sensors
ausGreen	contains green value of 16 sensors
ausBlue	contains blue value of 16 sensors
afLUX	will store the calculated LUX values of 16 sensors
CCT	will store the calculated CCT values of 16 sensors

2.8.4.3 int tcs_clearInt (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

clears the interrupt flag of 16 sensors.

Functions clears the interrupt flag of 16 sensors. The flag will be set if a color value has been exceeded, or the value read back from the sensor has fallen below a certain color value. Both settings can be set up in the sensors' registers.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Returns

0 : everything OK - conversions complete
1 : i2c-functions failed

2.8.4.4 int tcs_conversions_complete (unsigned char * *aucStatusRegister*)

checks if the ADCs for color measurement have already completed. Takes the status register as parameter.

Function checks if the sensors have already completed a color measuremend. In case the measurements are completed the ADCs can be safely read and used for further calculations. This can be checked by reading a special register of the sensor, the status register. If TCS3472_AVALID_BIT is set in this register, the ADCs have completed color measurements. If measurements are not completed, the return code can be used to determine which of the 16 sensor(s) failed.

Parameters

aucStatus↩ Register	holds the values of the status register for all 16 sensors

Return values

0	Succesful
>0	One or more sensors have not completed the conversion cycle yet, if the return code is 0b0000000000101100 for example, we have uncompleted conversions for sensor 3, sensor 4 and sensor 6

2.8.4.5 int tcs_exClear (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausClear,* unsigned char * *aucIntegrationtime*)

checks the clear color read back from the sensors have exceeded maximum clear.

Functions checks if the clear color read back from 16 sensors have exceeded a maximum clear level and are thus invalid. In case this maximum clear level is exceeded, one should consider lowering gain and/or integration time. If clear levels have been exceeded, the return code can be used to determine which of the 16 sensor(s) failed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
ausClear	clear values of the 16 sensors which will be checked for maximum clear level exceedings
auc↩ Integrationtime	current integration time setting of the 16 sensors - needed to check if maximum clear level has been exceeded

Return values

0	Succesful
>0	One or more sensors got saturated as they have reached the maximum amount in the clear data register, if the return code is 0b0000000000101100 for example, sensor 3, sensor 4 and sensor 6 got saturated

2.8.4.6 int tcs_getGain (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucGainSettings*)

reads the current gain setting of 16 sensors and stores them in an adequate buffer.

The function reads back the gain settings of 16 sensors. Refer to sensors' datasheet for further information about gain settings.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucGainSettings	pointer to buffer which will contain the gain settings of the 16 sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.7 int tcs_getIntegrationtime (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucIntegrationtime*)

reads the current integration time setting of 16 sensors and stores them in an adequate buffer.

The function reads back the integration time of 16 sensors. Refer to sensors' datasheet for further information about integration time settings.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
auc↩ Integrationtime	pointer to buffer which will store the integration time settings of the 16 sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.8 int tcs_identify (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned char * *aucReadbuffer*)

reads the ID-Register of 16 sensors and compares the values to expected values.

Expected ID for the TCS3472 is 0x44, the expected ID for the TCS3471 is 0x14

Parameters

ftdiA,ftdiB	pointer to ftdi_context
aucReadbuffer	stores the ID-values read back from the 16 sensors, must be able to hold 16 elements

Return values

0	Succesful
>0	Failed to identify one or more sensors, if the return code is 0b0000000000101100 for example, identification failed for sensor 3, sensor 4 and sensor 6
<0	USB or i2c errors occured, check return value for further information

2.8.4.9 int tcs_ON (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

turns 16 tcs3472 sensors on, releasing them from their sleep state.

Function wakes 16 sensors on in case they were put to sleep before. If sensors are already active this function has no effect.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.10 int tcs_readColor (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausColorArray,* tcs_color_t *color*)

reads back 4 color sets of 16 sensors - Red / Green / Blue / Clear.

Function reads 16-Bit color values of 16 sensors. The color will be specified by the input parameter tcs_color_t color.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

ausColorArray	will contain color value read back from 16 sensors
color	specifies the color to be read from the sensor (red, green, blue, clear)

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.11 int tcs_readColors (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* unsigned short * *ausClear,* unsigned short * *ausRed,* unsigned short * *ausGreen,* unsigned short * *ausBlue*)

reads back 4 colours and the content of the status register in one i2c command.

This function will read out the contents of the color registers of tcs3472 as words (2 Bytes per colour) and the content of the status register as one byte. The status register can be used in order to determine if color conversions had already completed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
ausClear	will contain color value read back from 16 sensors
ausRed	will contain color value read back from 16 sensors
ausGreen	will contain color value read back from 16 sensors
ausBlue	will contain color value read back from 16 sensors

```
@retval 0  Succesful
@retval <0 USB or i2c errors occured, check return value for further information
@retval >0 One or more sensors have not completed the conversion cycle yet,
           if the return code is 0b0000000000101100 for example, we have uncompleted conversions for sensor 3,
```

2.8.4.12 int tcs_setGain (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Gain_t *gain*)

sets the gain of 16 sensors.

Function sets the gain of 16 sensors. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
gain	gain to be sent to the sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.13 int tcs_setGain_x (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Gain_t *gain,* unsigned int *uiX*)

sets the gain of one sensor.

Function sets the gain of 1 sensors This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a greater gain factor, gain factor for bright LEDs can be low. Refer to the sensor's datasheet for further information about gain.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
gain	gain to be sent to the sensor
uiX	sensor which will get the new gain (0 ... 15)

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.14 int tcs_setIntegrationTime (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Integration_t *uiIntegrationtime*)

sets the integration time of 16 sensors at once.

Function sets the integration time of 16 sensors. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in enum tcs3472Integration_t.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
uiIntegrationtime	integration time to be sent to the sensors

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.15 int tcs_setIntegrationTime_x (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB,* tcs3472Integration_t *uiIntegrationtime,* unsigned int *uiX*)

sets the integration time of one sensor.

Function sets the integration time of one sensor. This setting can be used to capture both bright LEDs and dark LEDs. Whereas dark LEDs require a longer integration time, the integration time for bright LEDs can be low. Refer to the sensor's datasheet for calculating the content of the integration time register. A few common values have already been calculated and saved in enum tcs3472Integration_t.

Parameters

ftdiA,ftdiB	pointer to ftdi_context
uiIntegrationtime	integration time to be sent to the sensor
uiX	sensor which will get the new integration time (0 ... 15)

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.16 int tcs_sleep (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

sends 16 sensors to sleep.

Function sends 16 color sensors to sleep state.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information

2.8.4.17 int tcs_waitForData (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

checks if the ADCs for color measurement have already completed.

Function checks if the sensors have already completed a color measuremend. In case the measurements are completed the ADCs can be safely read and the datasets are valid. This can be checked by reading a special register of the sensor, the status register. If TCS3472_AVALID_BIT is set in this register, the ADCs have completed color measurements. If measurements are not completed, the return code can be used to determine which of the 16 sensor(s) failed.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
>0	One or more sensors have not completed the conversion cycle yet, if the return code is 0b0000000000101100 for example, we have uncompleted conversions for sensor 3, sensor 4 and sensor 6
<0	USB or i2c errors occured, check return value for further information

2.8.4.18 int tcs_wakeUp (struct ftdi_context * *ftdiA,* struct ftdi_context * *ftdiB*)

wakes up 16 color sensors.

Function wakes 16 color sensors from sleep state.

Parameters

ftdiA,ftdiB	pointer to ftdi_context

Return values

0	Succesful
<0	USB or i2c errors occured, check return value for further information