SOLUS Probe SDK

SOLUS

Version 1.0.0

Software Development Kit

Manual

February, 2019

Contents

Chapter 1

SOLUS Software Development Kit (SOLUS-SDK)

Software Development Kit (SDK) for the SOLUS probe. This software allows the communication between a Windows PC and the SOLUS probe. It must be included in the user software as a DLL, which provides all the functions to properly configurate and operate the probe. In order to execute a program which links to the SDK library, the following files are required:

SOLUS_SDK.dll	Software development kit library

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Module Documentation

4.1 SOLUS-SDK macros

Macros

- #define MAX_SEQUENCE 384
- #define MAX_FRAMES 384∗8
- #define HISTOGRAM_BINS 128
- #define FRAME_SIZE_INT_HIST (2 + 4 + 192 + 2)
- #define FRAME_SIZE_INT (2 + 4 + 2)
- #define N_LD 4
- #define N_OPTODE 4
- #define N_PIXEL 1728
- #define EEPROM_SIZE 4096

4.1.1 Detailed Description

Macro definitions for SOLUS-SDK. DO NOT CHANGE.

4.1.2 Macro Definition Documentation

4.1.2.1 MAX_SEQUENCE

```
#define MAX_SEQUENCE 384
```

Max Sequence Lenght.

4.1.2.2 MAX_FRAMES

```
#define MAX_FRAMES 384*8
```

Max frames.

4.1.2.3 HISTOGRAM_BINS

```
#define HISTOGRAM_BINS 128
```

Number of histogram bins.

4.1.2.4 FRAME_SIZE_INT_HIST

```
#define FRAME_SIZE_INT_HIST (2 + 4 + 192 + 2)
```

Frame size histogram mode.

4.1.2.5 FRAME_SIZE_INT

```
#define FRAME_SIZE_INT (2 + 4 + 2)
```

Frame size intensity mode.

4.1.2.6 N_LD

```
#define N_LD 4
```

Number of laser driver chip per optode.

4.1.2.7 N_OPTODE

```
#define N_OPTODE 4
```

Number of optode. TODO: change to 8!!!!

Examples

SOLUS_Example.c.

4.1.2.8 N_PIXEL

```
#define N_PIXEL 1728
```

Number of GSIPM pixels.

4.1.2.9 EEPROM_SIZE

```
#define EEPROM_SIZE 4096
```

EEPROM size.

4.2 SOLUS-SDK custom Types and structures

Data Structures

- struct _LD_reg
- union LDs_registers
- struct LD_parameters
- struct _GSIPM_reg
- union GSIPM_config_reg
- struct GSIPM_parameters
- struct Control_params
- struct _Sequence_Line_LL
- struct Sequence
- union Sequence_LL
- struct _LD_status
- union LDs_status
- struct Optode_analog_acq
- struct Control_analog_acq
- struct LDs_analog
- struct Frame

Typedefs

- typedef UINT32 DCRmap[N_OPTODE][N_PIXEL]
- typedef Frame Acquired_data[MAX_FRAMES]
- typedef UINT16 CalMap[N_PIXEL]
- typedef BOOLEAN Opt_Present[N_OPTODE]
- typedef struct _SOLUS_H ∗ SOLUS_H
- typedef Acquired_data ∗ Data_H

Enumerations

- enum ADDRESS {
  OPTODE1 = 0, OPTODE2 = 1, OPTODE3 = 2, OPTODE4 = 3,
  OPTODE5 = 4, OPTODE6 = 5, OPTODE7 = 6, OPTODE8 = 7,
  CONTROL = 8 }
- enum SOLUS_Return {
  OK = 0, COMM_ERROR = -1, OUT_OF_MEMORY = -2, INVALID_POINTER = -3,
  COMM_TIMEOUT = -4, OUT_OF_RANGE = -5, INVALID_OP = -6, PROBE_ERROR = 7,
  FIRMWARE_NOT_COMPATIBLE = -8, OPTODE_NOT_PRESENT = -9, FIRMWARE_UPDATE_ERROR = -10, SEQUENCE_ALREADY_RUNNING = -11,
  NO_SEQUENCE_RUNNING = -12 }
- enum DataType { Intensity = 0, Int_Hist = 1 }

4.2.1 Detailed Description

Custom types used by the SDK. Must be used by the user to properly set/get data. Do not use raw data.

4.2.2 Data Structure Documentation

4.2.2.1 struct _LD_reg

Laser Driver register structure containing all registers for one LD chip (two laser drivers). Note that only few on them needs to be changed during normal use. They will be saved into the internal optode EEPROM and loaded on startup. If it is needed to change some of them, the user must load the actual values from the probe, save them into a local structure, modify only the required fields, and program the structure back to the optode (optionally saving it in the EEPROM).

Data Fields

UINT16	CH1_DELAY_F1: 10	Channel 1 Delay Fine.
UINT16	__pad0__: 6	Zero bits.
UINT8	CH1_DELAYC1: 4	Channel 1 Delay Coarse.
UINT8	__pad1__: 4	Zero bits.
UINT16	CH1_WIDTH_F1: 10	Channel 1 Width Fine.
UINT16	__pad2__: 6	Zero bits.
UINT16	CH1_WIDTH_C1: 5	Channel 1 Width Coarse.
UINT16	__pad3__: 11	Zero bits.
UINT16	CH1_IFINE1: 10	Channel 1 Current Fine.
UINT16	__pad4__: 6	Zero bits.
UINT8	CH1_ICOARSE1: 3	Channel 1 Current Coarse.
UINT8	__pad5__: 5	Zero bits.
UINT16	CH2_DELAY_F2: 10	Channel 2 Delay Fine.
UINT16	__pad6__: 6	Zero bits.
UINT8	CH2_DELAYC2: 4	Channel 2 Delay Coarse.
UINT8	__pad7__: 4	Zero bits.
UINT16	CH2_WIDTH_F2: 10	Channel 2 Width Fine.
UINT16	__pad8__: 6	Zero bits.
UINT16	CH2_WIDTH_C2: 5	Channel 2 Width Coarse.
UINT16	__pad9__: 11	Zero bits.
UINT16	CH2_IFINE2: 10	Channel 2 Current Fine.
UINT16	__pad10__: 6	Zero bits.
UINT8	CH2_ICOARSE2: 3	Channel 2 Current Coarse.
UINT8	__pad11__: 5	Zero bits.
UINT8	NETTLI: 1	Not TTL input mode.
UINT8	__pad12__: 3	Zero bits.
UINT8	CLKDIV: 2	Reference frequency divider.
UINT8	__pad13__: 2	Zero bits.
UINT8	DIVH: 3	Frequency divider for PLL hi.
UINT8	ENLP: 1	Enable Long Pulses.
UINT8	DIVL: 3	Frequency divider for PLL low.
UINT8	NEPLL: 1	Disable PLL.
UINT16	SYNCDF: 10	Sync delay fine.
UINT16	__pad14__: 6	Zero bits.
UINT8	SYNCDC: 4	Sync delay coarse.
UINT8	ESYNC: 1	Enable sync output.
UINT8	ETTLO: 1	Enable TTL output mode.
UINT8	__pad15__: 2	Zero bits.
UINT8	SELAD: 3	Select ADC/DAC source.
UINT8	ENADC: 1	Enable AD/DA converter.
UINT8	__pad16__: 2	Zero bits.
UINT8	ILIM_LSB: 2	Current limit LSB.
UINT8	ILIM_MSB: 8	Current limit MSB.
UINT8	CPONH: 1	Reference for charge pump block P.
UINT8	CPONL: 1	Reference for charge pump block P.
UINT8	__pad17__: 2	Zero bits.
UINT8	SDAREF: 2	Choose DA ref block I (hi or lo).

Data Fields

UINT8	__pad18__: 2	Zero bits.
UINT8	SYNCTR: 3	Sync trim in block S.
UINT8	__pad19__: 1	Zero bit.
UINT8	ETIMON: 1	Enable test current monitor block S.
UINT8	__pad20__: 3	Zero bits.
UINT8	CITR: 3	Trim output current in LDK (block O).
UINT8	__pad21__: 1	Zero bit.
UINT8	NCIEX: 1	Not CI voltage external (block O).
UINT8	NCIDIS: 1	Not CI disable, emergency shutdown (block O).
UINT8	__pad22__: 1	Zero bit.
UINT8	LBYP: 1	Bypass error check at startup.
UINT8	TSEL: 4	Test block selection 3bit.
UINT8	TM: 2	Testmode.
UINT8	__pad23__: 1	Zero bit.
UINT8	ENTAP: 1	Enable Test Access Port.
UINT8	CRC_CFG	CRC Checksum.

4.2.2.2 union LDs_registers

Laser Driver registers union containing registers of all LD chip (4 LD chips, 8 laser drivers in total). It is 32 x 4 bytes long.

Examples

SOLUS_Example.c.

Data Fields

struct _LD_reg	LD_reg[N_LD]	Array of Laser Driver registers
UINT8	bytes[sizeof(struct _LD_reg) ∗N_LD]	Array of 32 x 4 bytes

4.2.2.3 struct LD_parameters

Struct containing the most common laser parameters

Examples

SOLUS_Example.c.

Data Fields

UINT16	DELAY_F[8]	Laser Delay Fine array. Valid range 0..1023.
UINT8	DELAY_C[8]	Laser Delay Coarse array. Valid range 0..15.
UINT16	WIDTH_F[8]	Laser Width Fine array. Valid range 0..1023.
UINT8	WIDTH_C[8]	Laser Width Coarse array. Valid range 0..31.

Data Fields

UINT16	I_FINE[8]	Current Fine array. Valid range 0..1023.
UINT8	I_COARSE[8]	Current Coarse array. Valid range 0..7.
UINT16	SYNCD_F	Sync delay fine. Valid range 0..1023.
UINT8	SYNCD_C	Sync delay coarse. Valid range 0..15.

4.2.2.4 struct _GSIPM_reg

GSIPM register structure containing all registers for the GSIPM chip. Note that only few on them needs to be changed during normal use. They will be saved into the internal optode EEPROM and loaded on startup. If it is needed to change some of them, the user must load the actual values from the probe, save them into a local structure, modify the required fields, and program the structure back to the optode (optionally saving it in the EEPROM).

Data Fields

UINT8	VC2: 2	VC2 voltage
UINT8	VC1: 2	VC1 voltage
UINT8	VC1_ns: 2	VC1_ns voltage
UINT8	GATE_GEN_FORCE_EN: 1	Force gate generation on all cycle
UINT8	GATE_GEN_MONO_BYPASS: 1	Bypass gate generator monostable
UINT8	EN_QUADRANT_1: 1	Enable Quadrant 1
UINT8	EN_QUADRANT_2: 1	Enable Quadrant 2
UINT8	EN_QUADRANT_3: 1	Enable Quadrant 3
UINT8	EN_QUADRANT_4: 1	Enable Quadrant 4
UINT8	TDC_DITHER_CODE: 3	TDC dithering code
UINT8	TDC_DITHER_DISABLE: 1	Disable TDC dithering
UINT8	STOP	TDC stop position. Valid range 0..23.
UINT8	GATE_CLOSE	Gate closing position. Valid range 0..23.
UINT8	GATE_OPEN	Gate opening position. Valid range 0..23.

4.2.2.5 union GSIPM_config_reg

GSIPM registers union containing all registers for the GSIPM chip. It is 5 bytes long.

Examples

SOLUS_Example.c.

Data Fields

struct _GSIPM_reg	GSIPM_ref	GSIPM registers
UINT8	bytes[sizeof(struct _GSIPM_reg)]	5 bytes

4.2.2.6 struct GSIPM_parameters

Struct containing the most common GSIPM parameters

Examples

SOLUS_Example.c.

Data Fields

BOOLEAN	EN_QUADRANT_1	Enable Quadrant 1
BOOLEAN	EN_QUADRANT_2	Enable Quadrant 2
BOOLEAN	EN_QUADRANT_3	Enable Quadrant 3
BOOLEAN	EN_QUADRANT_4	Enable Quadrant 4
UINT8	STOP	TDC stop position in ns. Valid range: 0..23
UINT8	GATE_CLOSE	Gate closing position in ns. Valid range: 0..23
UINT8	GATE_OPEN	Gate opening position in ns. Valid range: 0..23

4.2.2.7 struct Control_params

Struct containing control PCb parameters

Examples

SOLUS_Example.c.

Data Fields

UINT16	LD_Voltage	Laser driver supply voltage
UINT16	SPAD_Voltage	SPAD supply voltage
UINT16	GSIPM3v3_Voltage	GSIPM 3.3V supply voltage

4.2.2.8 struct _Sequence_Line_LL

Low level Sequence line structure containing one line of the measurement sequence. It is 6 bytes long.

Examples

SOLUS_Example.c.

Data Fields

UINT16	meas_time	Measurement time. If bit 16 is 0, bits 0-15 encode time with a 100us time bin, otherwise with a 10ms time bin.
UINT16	attenuation: 12	Attenuation, i.e. number of disabled SPADs.
UINT16	gate_dly_c: 4	Gate delay coarse
UINT16	gate_dly_f: 10	Gate delay fine
UINT16	laser_num: 6	Laser to fire

4.2.2.9 struct _Sequence_Line

High level Sequence line structure containing one line of the measurement sequence.

High level array containing all lines of the measurement sequence.

Data Fields

float	meas_time	Measurement time in seconds. Valid range is 100us..327s. Actual value is rounded with 100us precision up to 3s, then with 10ms precision up to 327s.
UINT16	attenuation	Attenuation, i.e. number of disabled SPADs. Valid range: 0..1727.
UINT8	gate_delay_coarse	Gate delay coarse. Valid range: 0..15.
UINT16	gate_delay_fine	Gate delay fine. Valid range: 0..1023.
UINT8	laser_num	Laser to fire. Valid range: 0..63.

4.2.2.10 union Sequence_LL

Low level Sequence union containing all lines of the measurement sequence.

Examples

SOLUS_Example.c.

Data Fields

struct _Sequence_Line_LL	Lines[MAX_SEQUENCE]	Array of Sequence Lines
UINT8	bytes[sizeof(struct _Sequence_Line_LL) ∗MAX_SEQUENCE]	6 x MAX_SEQUENCE bytes

4.2.2.11 struct _LD_status

Structure containing the status for a LD chip. It is 4 bytes long.

Data Fields

UINT16	ERR_VDD5: 1	5V supply voltage error
UINT16	ERR_VBG: 1	Bandgap reference error
UINT16	ERR_STUP: 1	Error during startup procedure
UINT16	ERR_ID: 1	Wrong ID (serial comm)
UINT16	ERR_IDRED: 1	ID redundancy error (RAM)
UINT16	ERR_OP: 1	Wrong opcode (serial comm)
UINT16	ERR_ADR: 1	Wrong address (serial comm)
UINT16	ERR_CRC: 1	CRC checksum error (RAM)
UINT16	ERR_LCKL: 1	Error PLL low not locked
UINT16	ERR_LCKH: 1	Error PLL high not locked
UINT16	ERR_OVC: 1	Over current error
UINT16	ERR_OVT: 1	Over temperature error

Data Fields

UINT16	ERR_PULSE: 1	Pulse width error
UINT16	ERR_CLKI: 1	Internal 10M clock error
UINT16	ERR_FFBIH: 1	Internal PLL timing error hi
UINT16	ERR_FFBIL: 1	Internal PLL timing error low
UINT16	ERR_CI: 1	Output current error
UINT16	ERR_CHANN: 1	No channel selected

4.2.2.12 union LDs_status

Array of union with status of all Laser Driver chips. It is 4 x 4 bytes long.

Examples

SOLUS_Example.c.

Data Fields

struct _LD_status	status[N_LD]	LD status structure
UINT32	u32[N_LD]	4 x 4 bytes

4.2.2.13 struct Optode_analog_acq

Structure containing the analog acquisitions for an optode.

Examples

SOLUS_Example.c.

Data Fields

INT16	gsipmSPADcurrent	GSIPM SPAD current in uA
INT16	gsipmCoreCurrent	GSIPM core current in hundreds of uA
INT16	laserCurrent	Laser current in tens of uA
UINT16	gsipmSPADvoltage	GSIPM SPAD voltage in mV
UINT16	gsipmCoreVoltage	GSIPM core voltage in mV
UINT16	laserVoltage	Laser voltage in mV
INT16	picTemperature	PIC Temperature in 0.01 C
UINT16	gsipmTemperature	GSIPM Temperature (in code for now)
UINT16	bandgap	Bandgap readout (code)

4.2.2.14 struct Control_analog_acq

Structure containing the analog acquisitions for the control board.

Examples

SOLUS_Example.c.

Data Fields

INT16	spadCurrent	SPAD current in tens of uA
INT16	inputCurrent	Input current in hundreds of uA
UINT16	spadVoltage	SPAD voltage in mV
UINT16	inputVoltage	Input voltage in mV
UINT16	p5Volt	5V supply voltage in mV

4.2.2.15 struct _LD_Analog

Structure containing the analog acquisitions for a LD chip

Array of structures containing the analog acquisitions for all the laser drivers of an optode.

Data Fields

UINT16	ILDK	Laser driver current at LDK
UINT16	VCI	Voltage at the driver stage
UINT16	V18	Generated 1.8V supply
UINT16	VDD	Supply voltage
UINT16	Temp	Chip Temperature

4.2.2.16 struct Frame

Structure containing one acquisition frame.

Examples

SOLUS_Example.c.

Data Fields

UINT32	intensity_data	Total number of detection during an acquision frame
UINT16	histogram_data[HISTOGRAM_BINS]	Complete histogram of the acquisition frame
UINT16	Area_ON	Number of enabled SPAD during the acquisition
ADDRESS	Optode	Optode to which the frame refers
UINT8	Status	Status of the probe during acquisition

4.2.3 Typedef Documentation

4.2.3.1 DCRmap

```
typedef UINT32 DCRmap[N_OPTODE][N_PIXEL]
```

Bidimensional array containing the DCR of the detectors.

Examples

SOLUS_Example.c.

4.2.3.2 Acquired_data

```
typedef Frame Acquired_data[MAX_FRAMES]
```

Array containing a full measurement. Each member of the array is an acquisition frame, according to the programmed sequence. Each line of the sequence corresponds to N frames, where N is the number of installed optodes (typically 8).

Examples

SOLUS_Example.c.

4.2.3.3 CalMap

```
typedef UINT16 CalMap[N_PIXEL]
```

Array containing the activation list for calibration. The array entry is the address of the SPAD and the array index is the activation order.

4.2.3.4 Opt_Present

```
typedef BOOLEAN Opt_Present[N_OPTODE]
```

Array of boolean showing which optode are actually installed and working in the probe.

Examples

SOLUS_Example.c.

4.2.3.5 SOLUS_H

```
typedef struct _SOLUS_H* SOLUS_H
```

Handle to the SOLUS structure.

4.2.3.6 Data_H

```
typedef Acquired_data* Data_H
```

Handle to the SOLUS data structure.

4.2.4 Enumeration Type Documentation

4.2.4.1 ADDRESS

```
enum ADDRESS
```

Enum type for the optode/control address.

Enumerator

OPTODE1	Optode 1 address
OPTODE2	Optode 2 address
OPTODE3	Optode 3 address
OPTODE4	Optode 4 address
OPTODE5	Optode 5 address
OPTODE6	Optode 6 address
OPTODE7	Optode 7 address
OPTODE8	Optode 8 address
CONTROL	Control address

4.2.4.2 SOLUS_Return

```
enum SOLUS_Return
```

Error codes returned by the SOLUS functions.

Enumerator

OK	The function returned successfully.
COMM_ERROR	Communication error. Check connections.
OUT_OF_MEMORY	Not enough memory to allocate the structure.
INVALID_POINTER	An invalid pointer was passed as parameter.
COMM_TIMEOUT	Timeout during communication.
OUT_OF_RANGE	Parameters out of range.
INVALID_OP	The required function can not be executed.
PROBE_ERROR	Probe returned an error code.
FIRMWARE_NOT_COMPATIBLE	Firmware not compatible.

Enumerator

OPTODE_NOT_PRESENT	Optode not present or not working.
FIRMWARE_UPDATE_ERROR	Error during firmware update.
SEQUENCE_ALREADY_RUNNING	A sequence is already running.
NO_SEQUENCE_RUNNING	A sequence has not been started yet.

4.2.4.3 DataType

```
enum DataType
```

Types of acquisition data.

Enumerator

Intensity	Only intensity is acquired
Int_Hist	Intensity and full histogram are acquired

4.3 Constructror, destructor, error handling

Functions

- SOLUS_Return SOLUS_Constr (SOLUS_H ∗SOLUS, Opt_Present ∗OptList)
- SOLUS_Return SOLUS_Destr (SOLUS_H SOLUS)
- void PrintErrorCode (const char ∗FunName, SOLUS_Return retcode)

4.3.1 Detailed Description

Functions to construct and destruct SOLUS objects and for error handling

4.3.2 Function Documentation

4.3.2.1 SOLUS_Constr()

```
SOLUS_Return SOLUS_Constr (
            SOLUS_H * SOLUS,
            Opt_Present * OptList )
```

SOLUS Constructor. Allocates a memory block to contain data for a SOLUS probe and opens the communication.

Parameters

SOLUS	Pointer to SOLUS handle.
OptList	Pointer to the array in which the constructor will save which optode is present and working.

Returns

OK The structure was successfully created and communication opened.
INVALID_POINTER A not empty pointer was passed.
OUT_OF_MEMORY Not enough memory to allocate the structure.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.3.2.2 SOLUS_Destr()

```
SOLUS_Return SOLUS_Destr (
            SOLUS_H SOLUS )
```

SOLUS Destructor. Deallocates the memory and closes the communication.

Parameters

SOLUS	SOLUS handle

Returns

OK The structure was successfully deallocated and communication closed.
INVALID_POINTER An empty SOLUS handle was passed.

Examples

SOLUS_Example.c.

4.3.2.3 PrintErrorCode()

```
void PrintErrorCode (
            const char * FunName,
            SOLUS_Return retcode )
```

Print an error message. All the SDK functions return an error code to inform the user whether the issued command was successfully executed or not. This function prints out the description of the error code on the standard output.

Parameters

FunName	Additional text to define the warning/error. Usually the name of the calling function is provided.
retcode	Error code returned by a SDK command

4.4 Set methods

Functions

- SOLUS_Return SOLUS_SetOptodeRegs (SOLUS_H solus, ADDRESS optode, LDs_registers ∗LD_Registers, GSIPM_config_reg ∗GSIPM_Registers)
- SOLUS_Return SOLUS_SetOptodeParams (SOLUS_H solus, ADDRESS optode, LD_parameters LD_Parameters, GSIPM_parameters GSIPM_parameters)
- SOLUS_Return SOLUS_SetCalibrationMap (SOLUS_H solus, ADDRESS optode, CalMap ∗data)
- SOLUS_Return SOLUS_SetSequenceLL (SOLUS_H solus, Sequence_LL ∗sequence)
- SOLUS_Return SOLUS_SetSequence (SOLUS_H solus, Sequence ∗sequence)
- SOLUS_Return SOLUS_SetLaserFrequency (SOLUS_H solus, UINT32 Frequency)

4.4.1 Detailed Description

Functions to set parameters of the SOLUS probe.

4.4.2 Function Documentation

4.4.2.1 SOLUS_SetOptodeRegs()

```
SOLUS_Return SOLUS_SetOptodeRegs (
            SOLUS_H solus,
            ADDRESS optode,
            LDs_registers ∗ LD_Registers,
            GSIPM_config_reg ∗ GSIPM_Registers )
```

Set optode registers. Sets the registers for a specific optode. The function requires pointers to user space structures containing laser drivers and GSIPM registers. When changing optode parameters, a Read-Modify-Write sequence must be followed. This means that SOLUS_ReadOptodeRegs() and SOLUS_GetOptodeRegs() must be called before this function in order to receive in a user space variable the current settings. After that, the desired settings can be changed, and finally sent to the optode with this function.

Parameters

solus	SOLUS handle
optode	Address of the optode
LD_Registers	Pointer to a structure containing laser driver registers.
GSIPM_Registers	Pointer to a structure containing GSIPM registers.

Returns

OK Registers setting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
INVALID_POINTER An empty SOLUS handle or pointer to setting structures was passed.
OPTODE_NOT_PRESENT Optode not present or not working.
COMM_ERROR Communication error.

COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.4.2.2 SOLUS_SetOptodeParams()

```
SOLUS_Return SOLUS_SetOptodeParams (
            SOLUS_H solus,
            ADDRESS optode,
            LD_parameters LD_Parameters,
            GSIPM_parameters GSIPM_parameters )
```

Set optode parameters. Sets the parameter for a specific optode. The function requires pointers to user space structures containing laser drivers and GSIPM parameters. When changing optode parameters, a Read-Modify-↩ Write sequence must be followed. This means that SOLUS_GetOptodeParams() must be called before this function in order to receive in a user space variable the current settings. After that, the desired settings can be changed, and finally sent to the optode with this function.

Parameters

solus	SOLUS handle
optode	Address of the optode
LD_Parameters	Pointer to a structure containing laser driver parameters.
GSIPM_parameters	Pointer to a structure containing GSIPM parameters.

Returns

OK Parameters setting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
INVALID_POINTER An empty SOLUS handle or pointer to setting structures was passed.
OPTODE_NOT_PRESENT Optode not present or not working.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.4.2.3 SOLUS_SetCalibrationMap()

```
SOLUS_Return SOLUS_SetCalibrationMap (
            SOLUS_H solus,
            ADDRESS optode,
            CalMap * data )
```

Set calibration map. Sets the calibration map for a specific optode. The calibration is an array of N_PIXEL UINT16 values, containing the desired pixel activation order to be used by the internal calibration procedure.

Parameters

solus	SOLUS handle
optode	Address of the optode
data	Pointer to the array containing the calibration map.

Returns

OK Calibration map setting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
INVALID_POINTER An empty SOLUS handle or pointer to data was passed.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.4.2.4 SOLUS_SetSequenceLL()

```
SOLUS_Return SOLUS_SetSequenceLL (
            SOLUS_H solus,
            Sequence_LL * sequence )
```

Set measurement sequence, low level. Sets the measurement sequence (valid for all the optode) at low level. If the effective sequence lenght is less than MAX_SEQUENCE, all fields of unused sequence entries must be set to 0.

Parameters

solus	SOLUS handle
sequence	Pointer to a user space structure containing the measurement sequence.

Returns

OK Measurement sequence setting was successful.
INVALID_POINTER An empty SOLUS handle or pointer to sequence structure was passed.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.4.2.5 SOLUS_SetSequence()

```
SOLUS_Return SOLUS_SetSequence (
            SOLUS_H solus,
            Sequence * sequence )
```

Set measurement sequence, high level. Sets the measurement sequence (valid for all the optode) at high level. If the effective sequence lenght is less than MAX_SEQUENCE, all fields of unused sequence entries must be set to 0.

Parameters

solus	SOLUS handle
sequence	Pointer to a user space structure containing the measurement sequence.

Returns

OK Measurement sequence setting was successful.
INVALID_POINTER An empty SOLUS handle or pointer to sequence structure was passed.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.4.2.6 SOLUS_SetLaserFrequency()

```
SOLUS_Return SOLUS_SetLaserFrequency (
             SOLUS_H solus,
             UINT32 Frequency )
```

Setlaser frequency. Sets the laser frequency.

Parameters

solus	SOLUS handle
Frequency	New laser frequecny in Hz.

Returns

OK Laser frequency setting was successful.
INVALID_POINTER An empty SOLUS handle was passed.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.5 Get methods

Functions

- SOLUS_Return SOLUS_ReadOptodeRegs (SOLUS_H solus, ADDRESS optode)
- SOLUS_Return SOLUS_GetOptodeRegs (SOLUS_H solus, ADDRESS optode, LDs_registers ∗LD_↩ Registers, GSIPM_config_reg ∗GSIPM_Registers)
- SOLUS_Return SOLUS_GetOptodeParams (SOLUS_H solus, ADDRESS optode, LD_parameters ∗LD_↩ Parameters, GSIPM_parameters ∗GSIPM_parameters)
- SOLUS_Return SOLUS_ReadCalibrationMap (SOLUS_H solus, ADDRESS optode)
- SOLUS_Return SOLUS_GetCalibrationMap (SOLUS_H solus, ADDRESS optode, CalMap ∗data)
- SOLUS_Return SOLUS_GetSequenceLL (SOLUS_H solus, Sequence_LL ∗sequence)
- SOLUS_Return SOLUS_GetSequence (SOLUS_H solus, Sequence ∗sequence)
- SOLUS_Return SOLUS_ReadStatusOptode (SOLUS_H solus, ADDRESS optode)
- SOLUS_Return SOLUS_GetStatusOptode (SOLUS_H solus, ADDRESS optode, UINT16 ∗status, LDs_status ∗LD_Status)
- SOLUS_Return SOLUS_ReadStatusControl (SOLUS_H solus)
- SOLUS_Return SOLUS_GetStatusControl (SOLUS_H solus, UINT16 ∗status)
- SOLUS_Return SOLUS_ReadLaserFrequency (SOLUS_H solus)
- SOLUS_Return SOLUS_GetLaserFrequency (SOLUS_H solus, UINT32 ∗Frequency)

4.5.1 Detailed Description

Functions to get parameters from the SOLUS probe. Functions of the "Read" type are used to obtain the actual value for the probe and save it into the local probe structure. Functions of the "Get" type returns the locally stored value. This means that a sequence Read->Get must be followed to obtain the actual value for quantities still unknons or that can be changed by the probe itself, whereas a simple Get is enough (and recommended, to avoid useless communications), if the value can only be changed by the user.

4.5.2 Function Documentation

4.5.2.1 SOLUS_ReadOptodeRegs()

```
SOLUS_Return SOLUS_ReadOptodeRegs (
            SOLUS_H solus,
            ADDRESS optode )
```

Read registers from Optode. Reads all config registers for an optode an save them into the SOLUS object.

Parameters

solus	SOLUS handle
optode	Address of the optode

Returns

OK Parameters reading was successful.
INVALID_POINTER An empty SOLUS handle or pointer to sequence structure was passed.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.5.2.2 SOLUS_GetOptodeRegs()

```
SOLUS_Return SOLUS_GetOptodeRegs (
             SOLUS_H solus,
             ADDRESS optode,
             LDs_registers * LD_Registers,
             GSIPM_config_reg * GSIPM_Registers )
```

Get registers for an Optode. Gets the locally cached registers for an Optode. Must be proceeded by SOLUS_ReadOptodeRegs() if the actual and not cached values are desided (for instance to obtain the map loaded on startup from EEPROM, or when parameters have been changed by the optode itself during operation).

Parameters

solus	SOLUS handle
optode	Address of the optode
LD_Registers	Pointer to a structure to contain laser driver registers.
GSIPM_Registers	Pointer to a structure to contain GSIPM registers.

Returns

OK Registers getting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
INVALID_POINTER An empty SOLUS handle or pointer to setting structures was passed.

Examples

SOLUS_Example.c.

4.5.2.3 SOLUS_GetOptodeParams()

```
SOLUS_Return SOLUS_GetOptodeParams (
             SOLUS_H solus,
             ADDRESS optode,
             LD_parameters * LD_Parameters,
             GSIPM_parameters * GSIPM_parameters )
```

Get parameters for an Optode. Gets the currently applied parameters for an Optode.

Parameters

solus	SOLUS handle
optode	Address of the optode
LD_Parameters	Pointer to a structure to contain laser driver parameters.
GSIPM_parameters	Pointer to a structure to contain GSIPM parameters.

Returns

OK Parameters getting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
INVALID_POINTER An empty SOLUS handle or pointer to setting structures was passed.

4.5.2.4 SOLUS_ReadCalibrationMap()

```
SOLUS_Return SOLUS_ReadCalibrationMap (
             SOLUS_H solus,
             ADDRESS optode )
```

Read calibration map from Optode. Read the calibration map for an optode an save it into the SOLUS object.

Parameters

solus	SOLUS handle
optode	Address of the optode

Returns

OK Calibration map reading was successful.
INVALID_POINTER An empty SOLUS handle or pointer to sequence structure was passed.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.5.2.5 SOLUS_GetCalibrationMap()

```
SOLUS_Return SOLUS_GetCalibrationMap (
             SOLUS_H solus,
             ADDRESS optode,
             CalMap * data )
```

Set calibration map. Gets the locally stored calibration map for a specific optode. The calibration is an array of N_↩ PIXEL UINT16 values, containing the desired pixel activation order to be used by the internal calibration procedure. Must be proceeded by SOLUS_ReadCalibrationMap() if the actual and not cached values are desided (for instance to obtain the map loaded on startup from EEPROM).

Parameters

solus	SOLUS handle
optode	Address of the optode
data	Pointer to the array containing the calibration map.

Returns

OK Calibration map getting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
INVALID_POINTER An empty SOLUS handle or pointer to data was passed.

4.5.2.6 SOLUS_GetSequenceLL()

```
SOLUS_Return SOLUS_GetSequenceLL (
             SOLUS_H solus,
             Sequence_LL * sequence )
```

Get measurement Sequence, low level. Gets the locally stored measurement Sequence (for all optodes).

Parameters

solus	SOLUS handle
sequence	Pointer to a user space structure for storing the measurement sequence.

Returns

OK Measurement sequence setting was successful.
INVALID_POINTER An empty SOLUS handle or pointer to sequence structure was passed.

Examples

SOLUS_Example.c.

4.5.2.7 SOLUS_GetSequence()

```
SOLUS_Return SOLUS_GetSequence (
             SOLUS_H solus,
             Sequence * sequence )
```

Get measurement Sequence, high level. Gets the locally stored measurement Sequence (for all optodes).

Parameters

solus	SOLUS handle
sequence	Pointer to a user space structure for storing the measurement sequence.

Returns

OK Measurement sequence setting was successful.
INVALID_POINTER An empty SOLUS handle or pointer to sequence structure was passed.

Examples

SOLUS_Example.c.

4.5.2.8 SOLUS_ReadStatusOptode()

```
SOLUS_Return SOLUS_ReadStatusOptode (
             SOLUS_H solus,
             ADDRESS optode )
```

Read the status for an Optode. Reads the status for an optode an save it into the SOLUS object.

Parameters

solus	SOLUS handle
optode	Address of the optode

Returns

OK Status reading was successful.
INVALID_POINTER An empty SOLUS handle was passed.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.5.2.9 SOLUS_GetStatusOptode()

```
SOLUS_Return SOLUS_GetStatusOptode (
             SOLUS_H solus,
             ADDRESS optode,
             UINT16 * status,
             LDs_status * LD_Status )
```

Get the status for an Optode. Gets the locally stored status for an optode.

Parameters

solus	SOLUS handle
optode	Address of the optode
status	Pointer to an UINT16 variable to hold the optode global status.
LD_Status	Pointer to the structure to hold the laser driver status.

Returns

OK Status getting was successful.
OUT_OF_RANGE An invalid optode Address was passed to the function.
OPTODE_NOT_PRESENT Optode not present or not working.
INVALID_POINTER An empty SOLUS handle or pointer to structures was passed.

4.5.2.10 SOLUS_ReadStatusControl()

```
SOLUS_Return SOLUS_ReadStatusControl (
             SOLUS_H solus )
```

Read the status for control MCU. Reads the status for the control MCU an save it into the SOLUS object.

Parameters

solus	SOLUS handle

Returns

OK Status reading was successful.
INVALID_POINTER An empty SOLUS handle was passed.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.5.2.11 SOLUS_GetStatusControl()

```
SOLUS_Return SOLUS_GetStatusControl (
             SOLUS_H solus,
             UINT16 * status )
```

Get the status for control MCU. Gets the locally stored status for control MCU.

Parameters

solus	SOLUS handle
status	Pointer to an UINT16 variable to hold the status.

Returns

OK Status getting was successful.
INVALID_POINTER An empty SOLUS handle or pointer to structures was passed.

4.5.2.12 SOLUS_ReadLaserFrequency()

```
SOLUS_Return SOLUS_ReadLaserFrequency (
            SOLUS_H solus )
```

Read the actual laser frequency. Read actual value for laser frequency from probe MCU.

Parameters

solus	SOLUS handle

Returns

OK Laser frequency reading was successful.
INVALID_POINTER An empty SOLUS handle was passed.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

4.5.2.13 SOLUS_GetLaserFrequency()

```
SOLUS_Return SOLUS_GetLaserFrequency (
            SOLUS_H solus,
            UINT32 * Frequency )
```

Get the laser frequency. Gets the locally stored value for laser frequency.

Parameters

solus	SOLUS handle
Frequency	Pointer to an UINT32 variable to hold the frequency.

Returns

OK Laser frequency getting was successful.
INVALID_POINTER An empty SOLUS handle or pointer to structures was passed.

4.6 Acquisition methods

Functions

- SOLUS_Return SOLUS_StartSequence (SOLUS_H solus, DataType type, BOOLEAN autocal)
- SOLUS_Return SOLUS_StopSequence (SOLUS_H solus)
- SOLUS_Return SOLUS_QueryNLinesAvailable (SOLUS_H solus, UINT16 ∗NLines)
- SOLUS_Return SOLUS_GetMeasurement (SOLUS_H solus, Data_H ∗data, UINT16 NLines)
- SOLUS_Return SOLUS_StartDCRMeasurement (SOLUS_H solus, float IntegrationTime, UINT16 StartPixel, UINT16 StopPixel)
- SOLUS_Return SOLUS_GetDCRMeasurement (SOLUS_H solus, DCRmap ∗DCR)
- SOLUS_Return SOLUS_StopDCRSequence (SOLUS_H solus)

4.6.1 Detailed Description

Functions for acquiring data from the SOLUS probe.

4.6.2 Function Documentation

4.6.2.1 SOLUS_StartSequence()

```
SOLUS_Return SOLUS_StartSequence (
            SOLUS_H solus,
            DataType type,
            BOOLEAN autocal )
```

Start measurement sequence. Starts the measurement according to the programmed sequence and data type. Once the acquisition started data must be downloaded calling repeatedly the function SOLUS_GetMeasurement().

Parameters

solus	SOLUS handle
type	Specify the datatype of the measurement (only intensity or complete histogram and intensity)
autocal	Specify if autocalibration of attenuation should be performed before each line of the sequence. If FALSE, the value of attenuation specified in the sequence will be used.

Returns

OK Measurement started successfully.
INVALID_POINTER An empty SOLUS handle was passed.
INVALID_OP Acquisition already running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.6.2.2 SOLUS_StopSequence()

```
SOLUS_Return SOLUS_StopSequence (
             SOLUS_H solus )
```

Stop measurement sequence. Stops the running measurement. It must be called in any case before starting a new one.

Parameters

solus	SOLUS handle

Returns

OK Measurement stopped successfully.
INVALID_POINTER An empty SOLUS handle was passed.
INVALID_OP Acquisition not running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.6.2.3 SOLUS_QueryNLinesAvailable()

```
SOLUS_Return SOLUS_QueryNLinesAvailable (
             SOLUS_H solus,
             UINT16 * NLines )
```

Query Available Lines Gets the number of acquired sequence lines. Call this function before SOLUS_GetMeasurement() to know if all the desired lines has been measured.

Parameters

solus	SOLUS handle
NLines	Pointer to the number of acquired lines

Returns

OK Number of acquired lines succesfully read.
INVALID_POINTER An empty SOLUS of data handle was passed.
INVALID_OP Acquisition not running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.6.2.4 SOLUS_GetMeasurement()

```
SOLUS_Return SOLUS_GetMeasurement (
             SOLUS_H solus,
             Data_H * data,
             UINT16 NLines )
```

Get Measurement. Gets the desired number of sequence lines from a running measurement and provide to the user the address of the internal data structure. Each line of the sequence is composed by N frames, where N is the number of installed optodes (typically 8). Each frame rappresents an intensity or histogram measurement according to the programmed sequence. Data from the complete sequence may be acquired either with a single or multiple calls of this function, however the user must take care not to request in total more lines than the length of the currently programmed sequence. In order to avoid data corruption, it must be preceeded by SOLUS_QueryNLinesAvailable(), unless the user is sure that the measurement time is elapsed.

Parameters

solus	SOLUS handle
data	Pointer to the data structure handle
NLines	Number of sequence lines to be acquired. Accepted values: 1..384

Returns

OK Frames acquired successfully.
INVALID_POINTER An empty SOLUS of data handle was passed.
OUT_OF_RANGE NLines out of range.
INVALID_OP Acquisition not running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.6.2.5 SOLUS_StartDCRMeasurement()

```
SOLUS_Return SOLUS_StartDCRMeasurement (
             SOLUS_H solus,
             float IntegrationTime,
             UINT16 StartPixel,
             UINT16 StopPixel )
```

Start the DCR measurement. Starts the DCR measurement with the specified integration time and pixel range. Once the acquisition started data must be downloaded calling repeatedly the function SOLUS_GetDCRMeasurement().

Parameters

solus	SOLUS handle
IntegrationTime	Specify the integration time for the measurement in s.
StartPixel	Specity the first pixel to be acquired
StopPixel	Specity the last pixel to be acquired

Returns

OK Measurement started successfully.
INVALID_POINTER An empty SOLUS handle was passed.
INVALID_OP Acquisition already running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.6.2.6 SOLUS_GetDCRMeasurement()

```
SOLUS_Return SOLUS_GetDCRMeasurement (
             SOLUS_H solus,
             DCRmap * DCR )
```

Get DCR Measurement. Gets the DCR measurement and save data into the passed DCRmap. For each call of the function only the DCR for 1 pixel are acquired and thre corresponding entry in the DCRmap is polulated, thus the function must be called as many times as the required number of pixels, and it will scan automatically through them. Further call after the last pixel has been acquired will generate an error.

Parameters

solus	SOLUS handle
DCR	Pointer to the DCRmap array.

Returns

OK DCR acquired successfully.
INVALID_POINTER An empty SOLUS of data handle was passed.
INVALID_OP Acquisition not running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.6.2.7 SOLUS_StopDCRSequence()

```
SOLUS_Return SOLUS_StopDCRSequence (
             SOLUS_H solus )
```

Stop DCR measurement sequence. Stops the running DCR measurement. It must be called in any case befora starting a new sequence.

Parameters

solus	SOLUS handle

Returns

OK Measurement stopped successfully.
INVALID_POINTER An empty SOLUS handle was passed.
INVALID_OP Acquisition not running.
COMM_ERROR Communication error.
COMM_TIMEOUT Communication timeout.

Examples

SOLUS_Example.c.

4.7 Service methods

Functions

- SOLUS_Return SOLUS_TriggerCalibration (SOLUS_H solus)
- SOLUS_Return SOLUS_SaveEEPROM (SOLUS_H solus, ADDRESS address)
- SOLUS_Return SOLUS_ReadEEPROM (SOLUS_H solus, ADDRESS address, byte ∗data)
- SOLUS_Return SOLUS_ReadDiagOptode (SOLUS_H solus, ADDRESS Optode)
- SOLUS_Return SOLUS_GetDiagOptode (SOLUS_H solus, ADDRESS Optode, LDs_analog ∗LD_Analog, Optode_analog_acq ∗Optode_Analog)
- SOLUS_Return SOLUS_ReadDiagControl (SOLUS_H solus)
- SOLUS_Return SOLUS_GetDiagControl (SOLUS_H solus, Control_analog_acq ∗Control_Analog)
- SOLUS_Return SOLUS_ResetMCU (SOLUS_H solus, ADDRESS address)
- SOLUS_Return SOLUS_TriggerBootLoader (SOLUS_H solus, ADDRESS address, char ∗path)
- SOLUS_Return SOLUS_GetFWVersion (SOLUS_H solus, ADDRESS address, UINT16 ∗FW_ver)
- SOLUS_Return SOLUS_ReadMCU_ID (SOLUS_H solus, ADDRESS address)
- SOLUS_Return SOLUS_WriteFlags (SOLUS_H solus, ADDRESS address, UINT16 flags, UINT16 mask)
- SOLUS_Return SOLUS_PowerSupplyON (SOLUS_H solus, ADDRESS address, UINT16 config)
- SOLUS_Return SOLUS_PowerSupplyOFF (SOLUS_H solus, ADDRESS address, UINT16 config)
- SOLUS_Return SOLUS_LaserOFF (SOLUS_H solus)
- SOLUS_Return SOLUS_LaserON (SOLUS_H solus, ADDRESS address, UINT8 laser)
- SOLUS_Return SOLUS_SetControlParams (SOLUS_H solus, Control_params Params)
- SOLUS_Return SOLUS_GetControlParams (SOLUS_H solus, Control_params ∗Params)
- SOLUS_Return SOLUS_InitialSystemConfig (SOLUS_H solus, float temperature)

4.7.1 Detailed Description

Service functions for the SOLUS probe.

4.7.2 Function Documentation

4.7.2.1 SOLUS_TriggerCalibration()

```
SOLUS_Return SOLUS_TriggerCalibration (
             SOLUS_H solus )
```

Trigger calibration.

Parameters

solus	SOLUS handle

4.7.2.2 SOLUS_SaveEEPROM()

```
SOLUS_Return SOLUS_SaveEEPROM (
             SOLUS_H solus,
             ADDRESS address )
```

Save to EEPROM.

Parameters

solus	SOLUS handle
address	Address of the optode/control

4.7.2.3 SOLUS_ReadEEPROM()

```
SOLUS_Return SOLUS_ReadEEPROM (
             SOLUS_H solus,
             ADDRESS address,
             byte * data )
```

Read from EEPROM.

Parameters

solus	SOLUS handle
address	Address of the optode/control
data	Data read from EEPROM

4.7.2.4 SOLUS_ReadDiagOptode()

```
SOLUS_Return SOLUS_ReadDiagOptode (
             SOLUS_H solus,
             ADDRESS Optode )
```

Read optode diagnothic.

Parameters

solus	SOLUS handle
Optode	Address of the optode

Examples

SOLUS_Example.c.

4.7.2.5 SOLUS_GetDiagOptode()

```
SOLUS_Return SOLUS_GetDiagOptode (
            SOLUS_H solus,
            ADDRESS Optode,
            LDs_analog * LD_Analog,
            Optode_analog_acq * Optode_Analog )
```

Get optode diagnostic.

Parameters

solus	SOLUS handle
Optode	Address of the optode
LD_Analog	Analog readings from LDs
Optode_Analog	Analog readings from optode MCU

Examples

SOLUS_Example.c.

4.7.2.6 SOLUS_ReadDiagControl()

```
SOLUS_Return SOLUS_ReadDiagControl (
            SOLUS_H solus )
```

Read control MCU diagnostic.

Parameters

solus	SOLUS handle

Examples

SOLUS_Example.c.

4.7.2.7 SOLUS_GetDiagControl()

```
SOLUS_Return SOLUS_GetDiagControl (
            SOLUS_H solus,
            Control_analog_acq * Control_Analog )
```

Get control MCU diagnostic

Parameters

solus	SOLUS handle
Control_Analog	Analog readings from control MCU

Examples

SOLUS_Example.c.

4.7.2.8 SOLUS_ResetMCU()

```
SOLUS_Return SOLUS_ResetMCU (
             SOLUS_H solus,
             ADDRESS address )
```

Reset MCU

Parameters

solus	SOLUS handle
address	Address of the optode/control

4.7.2.9 SOLUS_TriggerBootLoader()

```
SOLUS_Return SOLUS_TriggerBootLoader (
             SOLUS_H solus,
             ADDRESS address,
             char * path )
```

Trigger bootloader programming.

Parameters

solus	SOLUS handle
address	Address of the optode/control
path	Path of the file (in bl2 format, generated by mplabx)

4.7.2.10 SOLUS_GetFWVersion()

```
SOLUS_Return SOLUS_GetFWVersion (
             SOLUS_H solus,
             ADDRESS address,
             UINT16 * FW_ver )
```

Get firmware version.

Parameters

solus	SOLUS handle
address	Address of the optode/control
FW_ver	Version of the current firmware

4.7.2.11 SOLUS_ReadMCU_ID()

```
SOLUS_Return SOLUS_ReadMCU_ID (
              SOLUS_H solus,
              ADDRESS address )
```

Read MCU ID.

Parameters

solus	SOLUS handle
address	Address of the optode/control

4.7.2.12 SOLUS_WriteFlags()

```
SOLUS_Return SOLUS_WriteFlags (
              SOLUS_H solus,
              ADDRESS address,
              UINT16 flags,
              UINT16 mask )
```

Write flags.

Parameters

solus	SOLUS handle
address	Address of the optode/control
flags	Flag word
mask	Mask word

Examples

SOLUS_Example.c.

4.7.2.13 SOLUS_PowerSupplyON()

```
SOLUS_Return SOLUS_PowerSupplyON (
             SOLUS_H solus,
             ADDRESS address,
             UINT16 config )
```

Turn ON power supplies.

Parameters

solus	SOLUS handle
address	Address of the optode/control
config	Configuration word

Examples

SOLUS_Example.c.

4.7.2.14 SOLUS_PowerSupplyOFF()

```
SOLUS_Return SOLUS_PowerSupplyOFF (
             SOLUS_H solus,
             ADDRESS address,
             UINT16 config )
```

Turn OFF power supplies.

Parameters

solus	SOLUS handle
address	Address of the optode/control
config	Configuration word

4.7.2.15 SOLUS_LaserOFF()

```
SOLUS_Return SOLUS_LaserOFF (
             SOLUS_H solus )
```

Turn OFF all laser diodes.

Parameters

solus	SOLUS handle

Examples

SOLUS_Example.c.

4.7.2.16 SOLUS_LaserON()

```
SOLUS_Return SOLUS_LaserON (
             SOLUS_H solus,
             ADDRESS address,
             UINT8 laser )
```

Turn ON one laser diode.

Parameters

solus	SOLUS handle
address	Address of the optode/control
laser	Number of the laser to be switched on

Examples

SOLUS_Example.c.

4.7.2.17 SOLUS_SetControlParams()

```
SOLUS_Return SOLUS_SetControlParams (
             SOLUS_H solus,
             Control_params Params )
```

Set control PCB parameters.

Parameters

solus	SOLUS handle
Params	Structure containing the parameters to be set

4.7.2.18 SOLUS_GetControlParams()

```
SOLUS_Return SOLUS_GetControlParams (
             SOLUS_H solus,
             Control_params * Params )
```

Get control PCB parameters.

Parameters

solus	SOLUS handle
Params	Pointer to the structure that will contain the parameters got from the probe.

4.7.2.19 SOLUS_InitialSystemConfig()

```
SOLUS_Return SOLUS_InitialSystemConfig (
             SOLUS_H solus,
             float temperature )
```

Compensate temperature.

Parameters

solus	SOLUS handle
temperature	Calibration temperature.

Chapter 5

File Documentation

5.1 SOLUS_SDK.h File Reference

Data Structures

- struct _LD_reg
- union LDs_registers
- struct LD_parameters
- struct _GSIPM_reg
- union GSIPM_config_reg
- struct GSIPM_parameters
- struct Control_params
- struct _Sequence_Line_LL
- struct Sequence
- union Sequence_LL
- struct _LD_status
- union LDs_status
- struct Optode_analog_acq
- struct Control_analog_acq
- struct LDs_analog
- struct Frame

Macros

- #define MAX_SEQUENCE 384
- #define MAX_FRAMES 384∗8
- #define HISTOGRAM_BINS 128
- #define FRAME_SIZE_INT_HIST (2 + 4 + 192 + 2)
- #define FRAME_SIZE_INT (2 + 4 + 2)
- #define N_LD 4
- #define N_OPTODE 4
- #define N_PIXEL 1728
- #define EEPROM_SIZE 4096

Typedefs

- typedef UINT32 DCRmap[N_OPTODE][N_PIXEL]
- typedef Frame Acquired_data[MAX_FRAMES]
- typedef UINT16 CalMap[N_PIXEL]
- typedef BOOLEAN Opt_Present[N_OPTODE]
- typedef struct _SOLUS_H ∗ SOLUS_H
- typedef Acquired_data ∗ Data_H

Enumerations

- enum ADDRESS {
  OPTODE1 = 0, OPTODE2 = 1, OPTODE3 = 2, OPTODE4 = 3,
  OPTODE5 = 4, OPTODE6 = 5, OPTODE7 = 6, OPTODE8 = 7,
  CONTROL = 8 }
- enum SOLUS_Return {
  OK = 0, COMM_ERROR = -1, OUT_OF_MEMORY = -2, INVALID_POINTER = -3,
  COMM_TIMEOUT = -4, OUT_OF_RANGE = -5, INVALID_OP = -6, PROBE_ERROR = 7,
  FIRMWARE_NOT_COMPATIBLE = -8, OPTODE_NOT_PRESENT = -9, FIRMWARE_UPDATE_ERROR = -10, SEQUENCE_ALREADY_RUNNING = -11,
  NO_SEQUENCE_RUNNING = -12 }
- enum DataType { Intensity = 0, Int_Hist = 1 }

Functions

- SOLUS_Return SOLUS_Constr (SOLUS_H ∗SOLUS, Opt_Present ∗OptList)
- SOLUS_Return SOLUS_Destr (SOLUS_H SOLUS)
- void PrintErrorCode (const char ∗FunName, SOLUS_Return retcode)
- SOLUS_Return SOLUS_SetOptodeRegs (SOLUS_H solus, ADDRESS optode, LDs_registers ∗LD_↩ Registers, GSIPM_config_reg ∗GSIPM_Registers)
- SOLUS_Return SOLUS_SetOptodeParams (SOLUS_H solus, ADDRESS optode, LD_parameters LD_↩ Parameters, GSIPM_parameters GSIPM_parameters)
- SOLUS_Return SOLUS_SetCalibrationMap (SOLUS_H solus, ADDRESS optode, CalMap ∗data)
- SOLUS_Return SOLUS_SetSequenceLL (SOLUS_H solus, Sequence_LL ∗sequence)
- SOLUS_Return SOLUS_SetSequence (SOLUS_H solus, Sequence ∗sequence)
- SOLUS_Return SOLUS_SetLaserFrequency (SOLUS_H solus, UINT32 Frequency)
- SOLUS_Return SOLUS_ReadOptodeRegs (SOLUS_H solus, ADDRESS optode)
- SOLUS_Return SOLUS_GetOptodeRegs (SOLUS_H solus, ADDRESS optode, LDs_registers ∗LD_↩ Registers, GSIPM_config_reg ∗GSIPM_Registers)
- SOLUS_Return SOLUS_GetOptodeParams (SOLUS_H solus, ADDRESS optode, LD_parameters ∗LD_↩ Parameters, GSIPM_parameters ∗GSIPM_parameters)
- SOLUS_Return SOLUS_ReadCalibrationMap (SOLUS_H solus, ADDRESS optode)
- SOLUS_Return SOLUS_GetCalibrationMap (SOLUS_H solus, ADDRESS optode, CalMap ∗data)
- SOLUS_Return SOLUS_GetSequenceLL (SOLUS_H solus, Sequence_LL ∗sequence)
- SOLUS_Return SOLUS_GetSequence (SOLUS_H solus, Sequence ∗sequence)
- SOLUS_Return SOLUS_ReadStatusOptode (SOLUS_H solus, ADDRESS optode)
- SOLUS_Return SOLUS_GetStatusOptode (SOLUS_H solus, ADDRESS optode, UINT16 ∗status, LDs_status ∗LD_Status)
- SOLUS_Return SOLUS_ReadStatusControl (SOLUS_H solus)
- SOLUS_Return SOLUS_GetStatusControl (SOLUS_H solus, UINT16 ∗status)
- SOLUS_Return SOLUS_ReadLaserFrequency (SOLUS_H solus)
- SOLUS_Return SOLUS_GetLaserFrequency (SOLUS_H solus, UINT32 ∗Frequency)
- SOLUS_Return SOLUS_StartSequence (SOLUS_H solus, DataType type, BOOLEAN autocal)
- SOLUS_Return SOLUS_StopSequence (SOLUS_H solus)

- SOLUS_Return SOLUS_QueryNLinesAvailable (SOLUS_H solus, UINT16 ∗NLines)
- SOLUS_Return SOLUS_GetMeasurement (SOLUS_H solus, Data_H ∗data, UINT16 NLines)
- SOLUS_Return SOLUS_StartDCRMeasurement (SOLUS_H solus, float IntegrationTime, UINT16 StartPixel, UINT16 StopPixel)
- SOLUS_Return SOLUS_GetDCRMeasurement (SOLUS_H solus, DCRmap ∗DCR)
- SOLUS_Return SOLUS_StopDCRSequence (SOLUS_H solus)
- SOLUS_Return SOLUS_TriggerCalibration (SOLUS_H solus)
- SOLUS_Return SOLUS_SaveEEPROM (SOLUS_H solus, ADDRESS address)
- SOLUS_Return SOLUS_ReadEEPROM (SOLUS_H solus, ADDRESS address, byte ∗data)
- SOLUS_Return SOLUS_ReadDiagOptode (SOLUS_H solus, ADDRESS Optode)
- SOLUS_Return SOLUS_GetDiagOptode (SOLUS_H solus, ADDRESS Optode, LDs_analog ∗LD_Analog, Optode_analog_acq ∗Optode_Analog)
- SOLUS_Return SOLUS_ReadDiagControl (SOLUS_H solus)
- SOLUS_Return SOLUS_GetDiagControl (SOLUS_H solus, Control_analog_acq ∗Control_Analog)
- SOLUS_Return SOLUS_ResetMCU (SOLUS_H solus, ADDRESS address)
- SOLUS_Return SOLUS_TriggerBootLoader (SOLUS_H solus, ADDRESS address, char ∗path)
- SOLUS_Return SOLUS_GetFWVersion (SOLUS_H solus, ADDRESS address, UINT16 ∗FW_ver)
- SOLUS_Return SOLUS_ReadMCU_ID (SOLUS_H solus, ADDRESS address)
- SOLUS_Return SOLUS_WriteFlags (SOLUS_H solus, ADDRESS address, UINT16 flags, UINT16 mask)
- SOLUS_Return SOLUS_PowerSupplyON (SOLUS_H solus, ADDRESS address, UINT16 config)
- SOLUS_Return SOLUS_PowerSupplyOFF (SOLUS_H solus, ADDRESS address, UINT16 config)
- SOLUS_Return SOLUS_LaserOFF (SOLUS_H solus)
- SOLUS_Return SOLUS_LaserON (SOLUS_H solus, ADDRESS address, UINT8 laser)
- SOLUS_Return SOLUS_SetControlParams (SOLUS_H solus, Control_params Params)
- SOLUS_Return SOLUS_GetControlParams (SOLUS_H solus, Control_params ∗Params)
- SOLUS_Return SOLUS_InitialSystemConfig (SOLUS_H solus, float temperature)
- SOLUS_Return SOLUS_GetArea (SOLUS_H solus, ADDRESS optode, UINT16 ∗area)
- SOLUS_Return SOLUS_SetArea (SOLUS_H solus, ADDRESS optode, UINT16 area)
- SOLUS_Return SOLUS_SetSingleSPAD (SOLUS_H solus, ADDRESS Optode, UINT16 spad_number)

5.1.1 Detailed Description

SOLUS probe software development kit. This C header contains all the functions and custom types needed to operate the SOLUS probe in user defined applications.

Chapter 6

Example Documentation

6.1 SOLUS_Example.c

```
//
// SOLUS_Example.cpp
//
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "SOLUS_SDK.h"
#include "defaults.h"
SOLUS_H solus = NULL;
LDs_registers *LDs_reg;
GSIPM_config_reg *GSIPMreg;
Opt_Present* OptList;
Data_H data;
DCRmap* DCR;
Frame frame;
Sequence *Seq;
Sequence_LL *Seq_LL;
Control_analog_acq ctrl_analog_acq;
Optode_analog_acq opt_analog_acq;
LDs_analog ld_analog;
LD_parameters *LDs_params;
GSIPM_parameters *GSIPMparams;
char *Test[] = {
        "Check structures size", //a
        "Get Analog acquisitions", //b
        "Program Optode N parameters", //c
        "Turn on laser N (turn off if no parameter)", //c
        "Sweep laser N current (requires parameter)", //d
};
void print_ctrl_analog_acq(Control_analog_acq c_aa) {
        printf("--- CONTROL ANALOG ACQUISITIONS -------------------------------\n");
        printf("        Voltage    Current\n");
        printf("Input    %4.2f V   %6.2f mA\n", (float)c_aa.inputVoltage/1000,
       (float)c_aa.inputCurrent/1e3*100);
        printf("SPAD     %4.2f V   %6.2f mA\n", (float)c_aa.spadVoltage/1000,
       (float)c_aa.spadCurrent/1e3*10);
        printf("+5V      %4.2f V\n", (float)c_aa.p5Volt/1000);
        printf("-----------------------------------------------------\n");
}
void print_opt_analog_acq(Optode_analog_acq o_aa, ADDRESS opt) {
        printf("--- OPTODE(%1d) ANALOG ACQUISITIONS -----------------------------\n", (UINT32)opt);
        printf("        Voltage    Current\n");
        printf("GSIPM    %4.2f V   %6.2f mA\n", (float)o_aa.gsipmCoreVoltage / 1000,
       (float)o_aa.gsipmCoreCurrent / 1e3 * 10);
        printf("SPAD     %4.2f V   %6.2f mA\n", (float)o_aa.gsipmSPADvoltage / 1000,
       (float)o_aa.gsipmSPADcurrent / 1e3);
        printf("LASER    %4.2f V   %6.2f mA\n", (float)o_aa.laserVoltage / 1000, (float)o_aa.laserCurrent /
       .1e3);
        printf("PIC Temperature %2.2f *C, R %i, BG %i\n", (float)o_aa.picTemperature / 100,
       o_aa.gsipmTemperature, o_aa.bandgap);
        printf("-----------------------------------------------------\n");
}
int main()
{
        BOOL loop = TRUE;
        int k, i;
        UINT16 NLines;
```

```
 int sel = 0;
 char c[10];
 SOLUS_Return a;
 INT8 i_c;
 LDs_reg = calloc(1, sizeof(LDs_registers));
 GSIPMreg = calloc(1, sizeof(GSIPM_config_reg));
 OptList = calloc(1, sizeof(Opt_Present));
 DCR = calloc(1, sizeof(DCRmap));
 Seq = calloc(1, sizeof(Sequence));
 Seq_LL = calloc(1, sizeof(Sequence_LL));
 LDs_params = calloc(1, sizeof(LD_parameters));
 GSIPMparams = calloc(1, sizeof(GSIPM_parameters));
 // ---- HW INIT --------------------------------------
 //constructor
 a=SOLUS_Constr(&solus, OptList);
 //get optode parameters
 SOLUS_GetOptodeRegs(solus, OPTODE2, LDs_reg, GSIPMreg);
 SOLUS_GetOptodeRegs(solus, OPTODE4, LDs_reg, GSIPMreg);
 while(loop) {
         // ---- MENU ------------------------------------

printf("\n********************************************************************************\n");
         printf(" SOLUS Test program\n");
         printf("********************************************************************************\n");
         for(i = 0; i < 5; i++) {
                 printf("\t%c) %s\n", i + ’a’, Test[i]);
         }
         printf("\tq) Quit\n> ");
         i_c = -1;
         do {
                 c[++i_c] = getchar();
         } while(c[i_c] != 10 && i_c<9);
         if(c[0] >= ’a’ && c[0] <= ’e’) {

printf("********************************************************************************\n");
                 printf("%s\n", Test[c[0] - ’a’]);

printf("********************************************************************************\n");
         }
         // ---- EXECUTE -----------------------------------
         switch(c[0]) {
                 case ’a’: // check size of structures
                         printf("Sequence (2304): %d\n", (int)sizeof(Sequence_LL));
                         printf("Line (6): %d\n", (int)sizeof(struct _Sequence_Line_LL));
                         printf("LDs registers (128): %d\n", (int)sizeof(LDs_registers));
                         printf("GSIPM registers (5): %d\n", (int)sizeof(GSIPM_config_reg));
                         printf("LDs status (16): %d\n", (int)sizeof(LDs_status));
                         printf("Data struct (820224): %d\n", (int)sizeof(Acquired_data));
                         printf("Data frame (267): %d\n", (int)sizeof(Frame));
                         printf("Control Parameters (6): %d\n", (int)sizeof(Control_params));
                         printf("DCRmap (55296): %d\n", (int)sizeof(DCRmap));
                         break;
                 case ’b’: // analog acquisitions
                         if(sscanf_s(c, "b%d", &sel)<=0) sel = 1;
                         for(k = 0; k < sel; k++) {
                                 SOLUS_ReadDiagControl(solus);
                                 SOLUS_GetDiagControl(solus, &ctrl_analog_acq);
                                 print_ctrl_analog_acq(ctrl_analog_acq);
                                 SOLUS_ReadDiagOptode(solus, OPTODE2);
                                 SOLUS_ReadDiagOptode(solus, OPTODE4);
                                 SOLUS_GetDiagOptode(solus, OPTODE2, &ld_analog, &opt_analog_acq);
                                 print_opt_analog_acq(opt_analog_acq, OPTODE2);
                                 SOLUS_GetDiagOptode(solus, OPTODE4, &ld_analog, &opt_analog_acq);
                                 print_opt_analog_acq(opt_analog_acq, OPTODE4);
                                 Sleep(500);
                         }
                         sel = 0;
                         break;
                 case ’c’: // Program optode parameters
                         if(sscanf_s(c, "c%d", &sel)<=0) sel = 0;
                         memcpy(LDs_reg, &default_te173_regs, sizeof(LDs_registers));
                         memcpy(GSIPMreg, &default_GSIPM_regs, sizeof(GSIPM_config_reg));
                         LDs_reg->LD_reg[1].CH1_DELAYC1 = 2;
                         GSIPMreg->GSIPM_ref.GATE_CLOSE = 10;

                         SOLUS_SetOptodeRegs(solus, (ADDRESS)sel, LDs_reg, GSIPMreg);
                         //SOLUS_ReadOptodeRegs(solus, (ADDRESS)sel);
                         break;
                 case ’w’:
                         SOLUS_WriteFlags(solus, OPTODE2, 0x20, 0x20);
                         SOLUS_WriteFlags(solus, OPTODE4, 0x20, 0x20);
                         break;
                 case ’d’: // test single lasers
                         if(sscanf_s(c, "d%d", &sel) <= 0) sel = 0xFF;
                         if(sel != 0xFF) {
                                 /*LDs_params->DELAY_C[0] = 1;
                                 LDs_params->DELAY_F[0] = 1;
```

```
                                LDs_params->I_COARSE[7] = 6;
                                LDs_params->I_FINE[7] = 800;
                                LDs_params->SYNCD_F = 1;
                                LDs_params->WIDTH_F[7] = 1020;
                                LDs_params->WIDTH_C[7] = 6;
                                SOLUS_SetOptodeParams(solus, (ADDRESS)(sel » 3), *LDs_params,
*GSIPMparams); Sleep(5);*/
                                SOLUS_PowerSupplyON(solus, (ADDRESS)(sel » 3), 0x01);
                                SOLUS_LaserON(solus, (ADDRESS)(sel » 3), sel & 0x7);
                        } else {
                                SOLUS_LaserOFF(solus);
                                for(k = 0; k < N_OPTODE; k++) {
                                        if((*OptList)[k]) {
                                                SOLUS_PowerSupplyON(solus, (ADDRESS)(k), 0x02);
                                        }
                                }
                        }
                        break;
                case ’e’: // laser param sweep
                        if(sscanf_s(c, "e%d", &sel) > 0) {
                                memcpy(LDs_reg, &default_te173_regs, sizeof(LDs_registers));
                                memcpy(GSIPMreg, &default_GSIPM_regs, sizeof(GSIPM_config_reg));
                                SOLUS_PowerSupplyON(solus, (ADDRESS)(sel » 3), 0x01); Sleep(2);
                                printf("\n");
                                for(k = 6; k > 0; k-) {
                                        if(sel % 2) {
                                                LDs_reg->LD_reg[(sel % 8) » 1].CH2_WIDTH_C2 = k;
                                        } else {
                                                LDs_reg->LD_reg[(sel % 8) » 1].CH1_WIDTH_C1 = k;
                                        }
                                        SOLUS_SetOptodeRegs(solus, (ADDRESS)(sel»3), LDs_reg,
GSIPMreg); Sleep(2);
                                        SOLUS_LaserON(solus, (ADDRESS)(sel » 3), sel & 0x7);
                                        Sleep(1000);
                                        printf(".");
                                }
                                printf("\n");
                        }
                        break;
                case ’f’: // TBD
                        (*Seq)[0].meas_time = 1.0f;
                        (*Seq)[0].attenuation = 10;
                        (*Seq)[0].gate_delay_coarse = 1;
                        (*Seq)[0].gate_delay_fine = 50;
                        (*Seq)[0].laser_num = 14;
                        (*Seq)[1].meas_time = 5.0f;
                        (*Seq)[1].attenuation = 5;
                        (*Seq)[1].gate_delay_coarse = 2;
                        (*Seq)[1].gate_delay_fine = 40;
                        (*Seq)[1].laser_num = 13;
                        (*Seq)[2].meas_time = 3.0f;
                        (*Seq)[2].attenuation = 15;
                        (*Seq)[2].gate_delay_coarse = 3;
                        (*Seq)[2].gate_delay_fine = 60;
                        (*Seq)[2].laser_num = 12;
                        SOLUS_SetSequence(solus, Seq); Sleep(20);
                        SOLUS_GetSequenceLL(solus, Seq_LL);
                        (*Seq_LL).Lines[1].laser_num = 15;
                        SOLUS_SetSequenceLL(solus, Seq_LL); Sleep(200);
                        SOLUS_GetSequence(solus, Seq);
                        SOLUS_StartSequence(solus, Int_Hist, FALSE);
                        i = 0;
                        do {
                                i++;
                                SOLUS_QueryNLinesAvailable(solus, &NLines);
                        } while(NLines < 1);
                        SOLUS_GetMeasurement(solus, &data, 1);

                        if(*data) {
                                frame = (*data)[0];
                                printf("%d, %d\n", frame.intensity_data, i);
                                frame = (*data)[1];
                                frame = (*data)[2];
                        }
                        SOLUS_QueryNLinesAvailable(solus, &NLines);
                        SOLUS_StopSequence(solus);

                        break;
                case ’g’:

                        break;
                case ’h’:
                        (*DCR)[3][4] = 42;
                        SOLUS_StartDCRMeasurement(solus, 100, 1, 2);
                        SOLUS_GetDCRMeasurement(solus, DCR);
                        SOLUS_StopDCRSequence(solus);
                        SOLUS_GetDiagControl(solus, &ctrl_analog_acq);
```

```
                                break;
                        case ’q’:
                                loop = FALSE;
                                break;
                }
                for(i = 0; i < 10; i++) {
                        c[i] = ’\0’;
                }
        }
        // ---- Destructor --------------------------------------
        SOLUS_Destr(solus);
        solus = NULL;
        free(LDs_reg);
        free(GSIPMreg);
        free(OptList);
        free(DCR);
        free(Seq);
        free(Seq_LL);
        printf("Press ENTER to continue\n");
        //getchar();
        return 0;
}
```

Index