Sundance Multiprocessor Technology Limited **Software Design Specification**	Form : QCF54 Dated : 12 October 2000 Revision : 3 Approved : Mark Ainsworth

Product Name:	
Product Number:	SMT1036
Used On:	
Document Issue:	
Date:	28/06/13

Outline Description

Sundance Multiprocessor Technology Limited, Chiltern House, Waterside, Chesham, Bucks. HP5 1PS.

(Enquiries) Tel: +44 (0)1494 793167, Fax: +44 (0)1494 793168, e-mail: FabioA@Sundance.com

Revision History

Date	Changes Made	Issue	Initials
28/06/13	First revision	1.0	EW

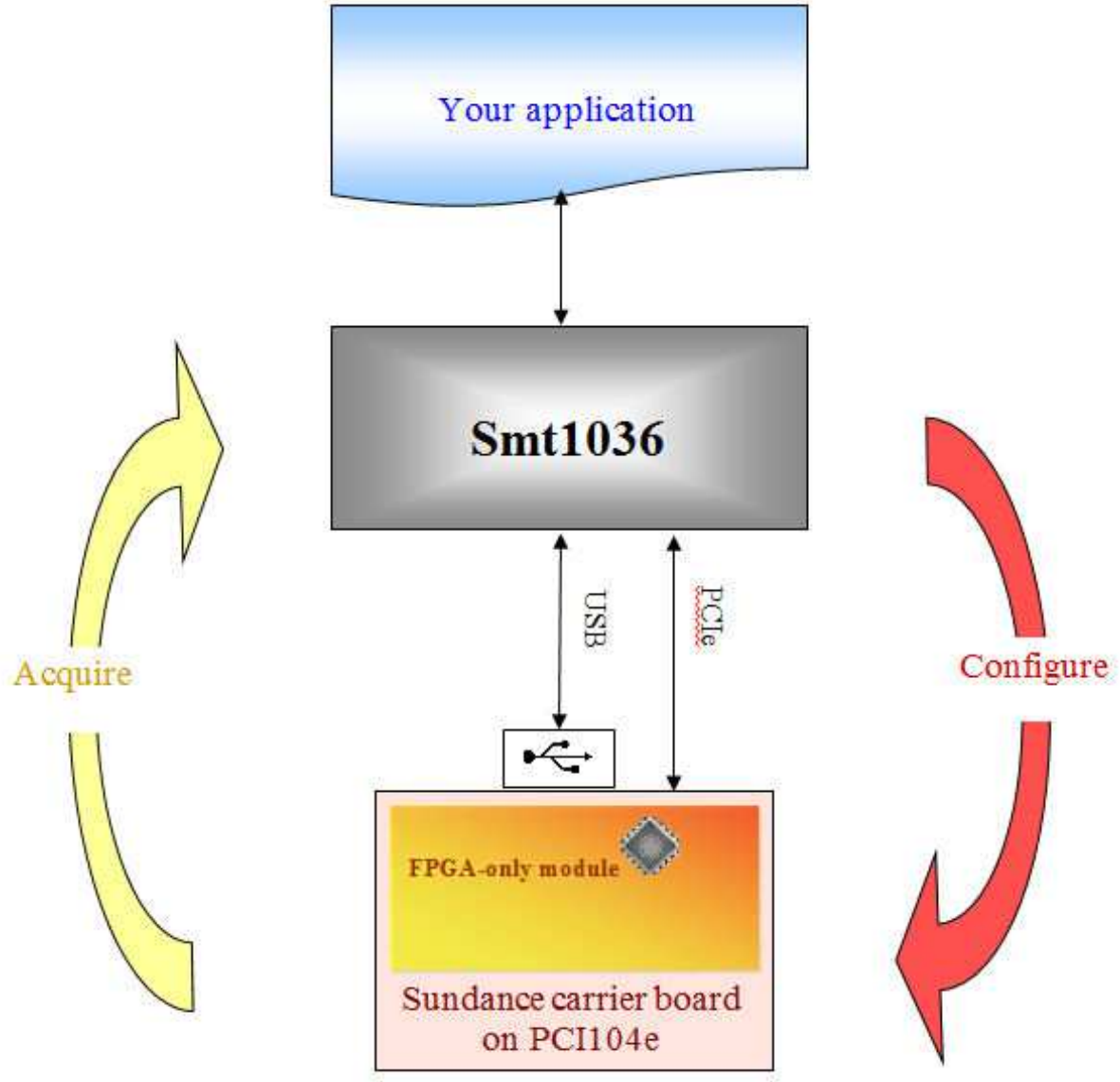
Table of Contents

1 Introduction

This section gives a brief description of the SMT1036 package.

The SMT1036 is an SDK for Linux. It is based on the Windows SDK SMT1026.

The SMT1036 is an efficient, ready to use, host side interface to Sundance FPGA-only module. It allows you to control the FPGA-only modules from the host as well as to exchange data between the host and the module on the first site of the Sundance carrier board.



The figure above shows the **SMT1036** forming the link between your application and the Sundance hardware in your system. The **SMT1036** hides the details of the device driver, allowing you to concentrate on the development process.

1.1 SMT1036 main features

This section lists the main characteristics of the SMT1036 package.

The SMT1036 SDK:

- Accelerate your development time by providing ready-to-use library of functions.
- Configure the FPGA from the Host.
- Transfer data from the FPGA-only module to the Host.
- Control the data acquisition on the daughter modules from the Host.
- Give you a basic framework for more complex custom systems.
- Provide you with C++ type interfaces to the FPGA-only modules.

1.2 Supported Hardware

The SMT1036 currently supports the following carrier boards:

Carrier board	Description	Functionality
SMT100	SLB carrier, Virtex 5, 512Mbytes DDR2, USB2, RS232, MicroSD/Transflash, 4-lane and 1-lane PCIe interface, 32-bit 33MHz PCI, cable PCIe connector	Full support
SMT101	SLB carrier, Virtex 5, 512Mbytes DDR2, USB2, RS232, MicroSD/Transflash, 4-lane and 1-lane PCIe interface, 8 channel 12-bit ADC, 32-bit 33 MHz PCI	Full support
SMT105	SLB carrier, Virtex 5, 512Mbytes DDR2, USB2, RS232, MicroSD/Transflash, 4-lane and 1-lane PCIe interface, 3 banks of QDRII memory 72 Mbit, 2x Fibre modules	Full support

The SMT1036 currently supports the following FPGA-only module:

Carrier board	Description
SMT350	Dual ADC/DAC module at 500MSPS
SMT351	1GB memory module
SMT370	Dual channel ADC/DAC
SMT381	Dual 14-bit DAC module at 1GSPS
SMT384	Quad 14-bit ADC module at 125MSPS
SMT390	Dual 12-bit ADC module at 210MSPs
SMT391	Dual 8-bit ADC module at 1GSPS
SMT941	4 channel 14-bit ADC at 250 MHz

2 Prerequisites

The language C++ is used for the software interfaces. Even if you are not familiar with C++, you should be able to find your way by referring to the samples. The samples are compiled and tested with Microsoft Visual Studio 2010, express edition.

The software can be distributed by CD or download.

The following **abbreviations** will be used through the all document:

FPGA	Field-Programmable Gate Array
CPLD	Complex PLDs
PCI	Peripheral Component Interconnect
PLD	Programmable Logic Device
SMT	Sundance Multiprocessor Technology Ltd.
TIM	Texas Instruments Module

3 Software implementation

This section describes the functionality of the SMT1036 package.

3.1 Interface Mechanism

The design makes use of a C++ style interface pointer to the hardware.

The SmtFPGA library exports functions that gather information about the installed boards and provide an interface pointer for later use.

To use the SMT1036, you need to:

- Obtain an interface pointer to the hardware by calling `SmtxxxOpen(unsigned int nIndex = 0)`.
- Use the interface pointer to call functions related to the hardware.

Example:

```
// Open the SMT391 library
IFSmt391 *pSmt391 = Smt391Open(0);
pSmt391->ResetTIMs();

// Configure the FPGA
pSmt391->ConfigureFPGA(BITSTREAM, QUICK_CONF);
```

3.2 Functions exported by SmtFPGA library

This section describes each of the functions exported by SmtFPGA library. These functions are described in the header file *IFSmtFPGA.h*.

- SmtFPGAOpen
- SmtFPGAClose
- Smt350Open
- Smt350Close
- Smt351Open
- Smt351Close
- Smt370Open
- Smt370Close
- Smt384Open
- Smt384Close
- Smt390Open
- Smt390Close
- Smt391Open
- Smt391Close
- Smt941Open
- Smt941Close

The following functions can be used to get information about the carrier board.

- SmtGetBoardCount
- SmtGetBoardIndex
- SmtGetBoardInfo
- SmtGetError

Functions to access the interfaces

SmtFPGAOpen

Prototype: `IFSmtFPGA * SmtFPGAOpen(unsigned int nIndex=0);`

Parameters: nIndex — Index of the carrier board on which the FPGA-only module is plugged in.

Return value: The return value is a pointer to an FPGA-only module interface.

Notes: This function obtains an interface to any FPGA-only module plugged in the first TIM site of a Sundance carrier board.

SmtFPGAClose

Prototype: `void SmtFPGAClose(IFSmtFPGA *p);`

Parameters: p — Pointer to the FPGA-only module interface.

Return value: None.

Notes: This function closes the FPGA-only module interface.

Smt350Open

Prototype: `IFSmt350 * Smt350Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which theSmt350 is plugged in.

Return value: The return value is a pointer to an Smt350 interface.

Notes: This function obtains an interface to an Smt350 plugged in the first TIM site of a Sundance carrier board.

Smt350Close

Prototype: `void Smt350Close(IFSmt350 *p);`

Parameters: p Pointer to theSmt350 interface.

Return value: None.

Notes: This function closes the Smt350 interface.

Smt351Open

Prototype: `IFSmt351 * Smt351Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which the Smt351 is plugged in.

Return value: The return value is a pointer to an Smt351 interface.

Notes: This function obtains an interface to an Smt351 plugged in the first TIM site of a Sundance carrier board.

Smt351Close

Prototype: `void Smt351Close(IFSmt351 *p);`

Parameters: p Pointer to the Smt351 interface.

Return value: None.

Notes: This function closes the Smt351 interface.

Smt370Open

Prototype: `IFSmt370 * Smt370Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which the Smt370 is plugged in.

Return value: The return value is a pointer to an Smt370 interface.

Notes: This function obtains an interface to an Smt370 plugged in the first TIM site of a Sundance carrier board.

Smt370Close

Prototype: `void Smt370Close(IFSmt370 *p);`

Parameters: p Pointer to the Smt370 interface.

Return value: None.

Notes: This function closes the Smt370 interface.

Smt384Open

Prototype: `IFSmt384 * Smt384Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which the Smt384 is plugged in.

Return value: The return value is a pointer to an Smt384 interface.

Notes: This function obtains an interface to an Smt384 plugged in the first TIM site of a Sundance carrier board.

Smt384Close

Prototype: `void Smt384Close(IFSmt384 *p);`

Parameters: p Pointer to the Smt384 interface.

Return value: None.

Notes: This function closes the Smt384 interface.

Smt390Open

Prototype: `IFSmt390 * Smt390Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which the Smt390 is plugged in.

Return value: The return value is a pointer to an Smt390 interface.

Notes: This function obtains an interface to an Smt390 plugged in the first TIM site of a Sundance carrier board.

Smt390Close

Prototype: `void Smt390Close(IFSmt390 *p);`

Parameters: p Pointer to the Smt390 interface.

Return value: None.

Notes: This function closes the Smt390 interface.

Smt391Open

Prototype: `IFSmt391 * Smt391Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which the Smt391 is plugged in.

Return value: The return value is a pointer to an Smt391 interface.

Notes: This function obtains an interface to an Smt391 plugged in the first TIM site of a Sundance carrier board.

Smt391Close

Prototype: `void Smt350Close(IFSmt391 *p);`

Parameters: p Pointer to the Smt391 interface.

Return value: None.

Notes: This function closes the Smt391 interface.

Smt941Open

Prototype: `IFSmt941 * Smt941Open(unsigned int nIndex=0);`

Parameters: nIndex Index of the carrier board on which the Smt941 is plugged in.

Return value: The return value is a pointer to an Smt941 interface.

Notes: This function obtains an interface to an Smt941 plugged in the first TIM site of a Sundance carrier board.

Smt941Close

Prototype: `void Smt941Close(IFSmt941 *p);`

Parameters: p Pointer to the Smt941 interface.

Return value: None.

Notes: This function closes the Smt941 interface.

Functions to get information about the carrier board

SmtGetBoardCount

Prototype: `DWORD SmtGetBoardCount(void);`

Parameters: None.

Return value: The return value is the number of carrier board found.

Notes: This function returns the number of Sundance carrier boards found in the system.

SmtGetBoardIndex

Prototype: `INT SmtGetBoardIndex(UINT nBaseAddress);`

Parameters: nBaseAddress Base address of the board to open.

Return value: The return value is the board index.

Notes: This function returns an index to the board at the base address nBaseAddress. If no board is found, the function returns -1.

SmtGetBoardInfo

Prototype: `SMTRet SmtGetBoardInfo(UINT nIndex, SMTBI& info);`

Parameters:

nIndex Index of the board to return information about.

info The information structure to populate.

Return value:	The return value is the board index.
Notes:	This function returns information about the carrier board specified by nIndex.

SmtGetError

Prototype: `const char * SmtGetError(SMTRet Error);`

Parameters: Error Error value to translate into a string.

Return value:	The return value is an error string.
Notes:	This function returns a string description of the error specified by Error.

3.3 Sundance Hardware Interface Description

Once an interface to the hardware has been obtained by calling `SmtXXXOpen(unsigned int nIndex=0)`, the interface allows you to access the FPGA-only module functions.

- General interface
- Smt350 interface
- Smt370 interface
- Smt351 interface
- Smt381 interface
- Smt384 interface
- Smt390 interface
- Smt391 interface
- Smt941 interface

3.3.1 General interface

The general interface can be used with any FPGA-only module plugged in a Sundance carrier board.

This section describes:

- the general functions
  - ConfigureFPGA
  - HostRead
  - HostWrite
  - HostCancel
  - WriteCtrlWord
  - StoreDataToFile
  - CallbackSet
  - CallbackGet
  - ResetTIMs
  - ResetBoard
  - GenSig
  - SetTimeout
  - GetTimeout

- the general structures
  - TEMP
  - SN
  - FILE_OPTIONS
- the general enumerated types
  - CHSEL
  - OUTPUT
  - VCXOTYPE
  - HOST_LINK
  - DATAFORMAT

The general functions

All the following functions can throw an SMTExc exception.

ConfigureFPGA

Prototype:

```
void ConfigureFPGA(const char *pBitstream, bool bQuickConf=false);
```

Parameters:

pBitstream	FPGA bitstream
bQuickConf	True: Reset the FPGA. The FPGA has to be fully configured at least once before using this option. False: Configure fully the FPGA. (default)

Return value: None.

Notes: This function configures the FPGA of the FPGA-only module plugged in the first TIM slot of the Sundance carrier board.

HostRead

Prototype:

```
void HostRead(void *pBuf, unsigned int nBytes, HOST_LINK hostlink=cp);
```

Parameters:

pBuf	Buffer that receives the data.
nBytes	Size of the data to read.
hostlink	Communication link on which to read the data. Host comport per default. (See struct HOST_LINK).

Return value: None.

Notes: This function reads data from the carrier board.

HostWrite

Prototype:
```
void HostWrite(void *pBuf, unsigned int nBytes, HOST_LINK hostlink=cp);
```

Parameters:

pBuf	Buffer that contains the data to write.
nBytes	Size of the data to write.
hostlink	Communication link on which to write the data. Host comport per default. (See enum HOST_LINK).

Return value: None.

Notes: This function writes data to the carrier board.

HostCancel

Prototype:
```
void HostCancel(HOST_LINK hostlink=cp);
```

Parameters:

hostlink	Communication link on which to write the data. Host comport per default. (See enum HOST_LINK).

Return value: None.

Notes: This function cancels pending read/write operations.

WriteCtrlWord

Prototype:
```
void WriteCtrlWord(unsigned int nCtrl);
```

Parameters:

nCtrl	Control word to send.

Return value: None.

Notes: This function sends a control word to the FPGA-only module on the first TIM slot of a Sundance carrier board.

StoreDataToFile

Prototype:
```
void StoreDataToFile(const char *pcFile, void *pBuf, unsigned int nBytes, unsigned int nNoOfBit, FILE_OPTIONS *=0);
```

Parameters:

pcFile	Name of the file where the data will be saved.
pBuf	Buffer containing the data.

	nBytes	Size of the data.
	nNoOfBit	Number of relevant bits per word. This value is used to mask each word saved in the file.
	pFileOpt	Structure which contains the extra file options (append data, add date and time). (See struct FILE_OPTIONS).

Return value: None.

Notes: This function stores data into a file.

CallbackSet

Prototype: `void CallbackSet(SMTFPGACALLBACK pf, void *user);`

Parameters:

pf	Callback function.
user	User specified callback value.

Return value: None.

Notes: This function sets the function that is to be called when words are sent to the FPGA module.

CallbackGet

Prototype: `void CallbackGet(SMTFPGACALLBACK pf, void *user);`

Parameters:

pf	Callback function.
user	User specified callback value.

Return value: None.

Notes: This function gets the function that is to be called when words are sent to the FPGA module.

ResetTIMs

Prototype: `void ResetTIMs(void);`

Parameters: None.

Return value: None.

Notes: This function resets the TIMs on the carrier board.

ResetBoard

Prototype: `void ResetBoard(void);`

Parameters: None.

Return value: None.

Notes: This function resets the carrier board.

GenSig

Prototype:
```
void GenSig(SIGGEN type, int nAmp, float fSamplingFreq, float fSigFreq, unsigned int *pBuf, int nBytes);
```

Parameters:

type	Type of signal to generate (See SIGGEN).
nAmp	Signal amplitude.
fSamplingFreq	Sampling frequency.
fSigFreq	Signal frequency.
pBuf	Buffer for the signal data.
nBytes	Size of the buffer in bytes.

Return value: None.

Notes: This function generates a signal.

SetTimeout

Prototype: `void SetTimeout(unsigned int n);`

Parameters:

n	Value of the timeout in ms. If 0 the timeout is not set and the comport and RSL functions will never return if they fail.

Return value: None.

Notes: This function changes the value of the timeout for the comport and RSL functions. Unless really necessary the user is not advised to use this function.

GetTimeout

Prototype: `unsigned int GetTimeout(void);`

Parameters: None.

Return value: The current value of the timeout.

Notes: This function returns the current value of the timeout for the comport and RSL functions.

The general structures

struct TEMP	
`unsigned int nAir`	Temperature measured on the base module such as an SMT338VP.
`unsigned int nFPGA`	Temperature measured on the FPGA of the base module.
`unsigned int nDaughter_Air`	Temperature measured on the daughter module.
`unsigned int nDaughter_ADC`	Temperature measured on the ADCs of the daughter module.

struct SN	
`unsigned int nNoA`	Serial number A of the base module.
`unsigned int nNoB`	Serial number B of the base module.
`unsigned int nNoC`	Serial number C of the base module.
`unsigned int nNoD`	Serial number D of the base module.
`unsigned int nDaughterNoA`	Serial number A of the daughter module.
`unsigned int nDaughterNoB`	Serial number B of the daughter module.
`unsigned int nDaughterNoC`	Serial number C of the daughter module.
`unsigned int nDaughterNoD`	Serial number D of the daughter module.

struct FILE_OPTIONS	
`bool bAppend`	True = append data at the end of the file.

	False = overwrite the file. The default value is false.
bool bDateStamp	True = add date and time before writing the data. False = don't add the date and time inside the file. The default value is false.

The general enumerated types

Enumerated types	Description

enum **CHSEL** {
channela = 1,
channelb = 2,
channelab= 3
};
	Selects the active channel.

enum **OUTPUT** {
normal = 0,
zeros = 1,
ones = 2,
test = 3
};
	Selects the type of the output samples.

enum **VCXOTYPE** {
vcxo100 = 0,
vcxo245 = 1
};
	Selects the VCXO installed on the FPGA module: 100Mhz or 245.76 MHz

enum **HOST_LINK** {
cp = 0,
rsl = 1
};
	The host application can always communicate with the Sundance carrier boards through the host comport. But some carrier board offers other communication link like the RSL link. This enumerated type selects the host link used to communicate with the Sundance carrier board.

enum **DATAFORMAT** {
binary = 0,
complement2 = 1
};
	Selects the format of the output samples.

enum **SIGGEN** { sine = 0, triangle = 1, square = 2,	Selects the shape of the signal to generate.

```
    ramp_pos    =    3,
  ramp_neg = 4
};
```

3.3.2 SMT350 interface

The SMT350 interface can be used with a SMT350 plugged in a Sundance carrier board.

This section describes:

- the SMT350 functions
  - ResetDevices
  - ConfigureADCReg
  - ConfigureADCClock
  - Acquire
- the SMT350 structures
  - CLKMODE350
  - SETDAC
  - SETCLK
- the SMT350 enumerated types
  - SMT350TYPE
  - SAMPLING_RATIO

The SMT350 functions

All the following functions can throw an SMTExc exception.

	ResetDevices
Prototype:	`void ResetDevices(SMT350TYPE smt350type=cdcm7005);`
Parameters:	smt350type — Select the SMT350 type: CDCM7005 or AD9510. (See enum SMT350TYPE).
Return value:	None.
Notes:	This function does a global reset of the SMT350 (ADCs, DACs, Clock…).

	ConfigureADCReg
Prototype:	`void ConfigureADCReg(OUTPUT outputADC, SETDAC *pSetDAC, SMT350TYPE smt350type, SETCLK *pSetClk, VCXOTYPE vcxotype=vcxo245);`
Parameters:	outputADC — Type of data the ADCs output. (See enum OUTPUT). pSetDAC — DAC parameters (gain and offset). (See struct SETDAC).

	smt350type	Select the SMT350 type: CDCM7005 or AD9510. (See enum SMT350TYPE).
	pSetClk	Clock parameters (sampling frequency ratio). (See struct SETCLK).
	vcxotype	Select the right VCXO used on the SMT350: 100MHz or 245.76 MHz. (See enum VCXOTYPE).

Return value: None.

Notes: This function configures the ADC, DAC and clock registers.

ConfigureADCClock

Prototype: `void ConfigureADCClk(CLKMODE350 *pClkMode);`

Parameters: pClkMode ADC clock mode parameters. (See struct CLKSMT350).

Return value: None.

Notes: This function configures the ADC clock.

Acquire

Prototype: `void Acquire(CLKMODE350 *pClkMode);`

Parameters: pClkMode ADC clock mode parameters. (See struct CLKSMT350).

Return value: None.

Notes: This function acquires data from the Smt350.

The SMT350 sructures

struct CLKMODE350	
`bool bExtRef`	true = external reference; false = on-board 10 MHz reference. The default value is false.
`bool bExtClk`	true = external clock; false = on-board VCXO. The default value is false.

`bool bTrigInvert`	true = trigger polarity channel A and B; false = Non-inverting. The default value is false.
`bool bTrigExt`	true = trigger selection channel A and B; false = internal trigger. The default value is false.
`CHSEL ChSel`	Channel selection. The default value is channelab.
`bool b2sComplement`	True = ADC A and B output 2's complement samples False = ADC A and B output binary samples. The default value is false.

struct SETDAC	
`unsigned int nOffsetA`	Offset DAC channel A. The default value is 0.
`unsigned int nGainA`	Gain DAC channel A. The default value is 0xFFF.
`unsigned int nOffsetB`	Offset DAC channel B. The default value is 0.
`unsigned int nGainB`	Gain DAC channel B. The default value is 0xFFF.

struct SETCLK	
`SAMPLING_RATIO ADCSamplingRatio`	Sampling frequency divider for the ADCs. The default value is div3.
`SAMPLING_RATIO DACSamplingRatio`	Sampling frequency divider for the DACs. The default value is div2.

The SMT350 enumerated types

Enumerated type	Description
`enum SMT350TYPE { ad9510 = 0, cdcm7005 = 1 };`	There are two types of SMT350. The default one with the component CDCM7005 and the one with the component AD9510. The right SMT350 type can be selected using this enumerated type.
`enum SAMPLING_RATIO`	This enumerated type selects the sampling frequency divider.

```
{
  div1  = 0,
  div2  = 1,
  div3  = 2,
  div4  = 3,
  div6  = 4,
  div8  = 5,
  div16 = 6
};
```

3.3.3 SMT351 interface

The SMT351 interface can be used with a SMT351 plugged a Sundance carrier board.

This section describes:

- the SMT351 functions
  - Acquire
  - ReadBack
  - FillMem

The SMT351 functions

All the following functions can throw an SMTExc exception.

Acquire

Prototype: `void Acquire(unsigned int nBytes, unsigned int nAddress);`

Parameters:

nBytes Number of bytes to acquire.

nAddress Memory address from where to acquire the data.

Return value: None.

Notes: This function prepares the SMT351 for the acquisition.

ReadBack

Prototype: `void ReadBack(void);`

Parameters: None.

Return None.

value:

Notes: This function starts the acquisition from the SMT351.

FillMem

Prototype:
```
void FillMem(unsigned int *pBuf, unsigned int nBytes);
```

Parameters:

pBuf	Buffer that contains the data to copy to the SMT351 memory.
nBytes	Number of bytes to copy to the SMT351 memory.

Return value: None.

Notes: This function acquires data from the SMT351.

3.3.4 SMT381 interface

The SMT381 interface can be used with a SMT381 plugged a Sundance carrier board.

This section describes:

- the SMT381 functions
  - ConfigureDACReg
  - ConfigureDACClock
  - SetTrigger
  - Reset
- the SMT381 enumerated types
  - DATAMODE

The SMT381 functions

All the following functions can throw an SMTExc exception.

ConfigureDACReg

Prototype:
```
void ConfigureDACReg(DATAMODE mode, unsigned short *pBuf, unsigned int nBurstSize);
```

Parameters:

mode	Select the data mode (See struct DATAMODE)..
pBuf	Buffer that contains the data to write to the DAC memory (only used when the direct mode is selected).
nBurstSize	Set the burst size for the DACs. Typically it will be the number of 14-bit samples divided by the number of samples per cycle

(NB_SAMPLE_CYCLE).

Return value: None.

Notes: This function configures the DAC registers of the SMT381.

ConfigureDACClock

Prototype:
```
unsigned int ConfigureADCClock(unsigned int nRf_SamplingFreq, bool bOnBoardClk);
```

Parameters:

nRf_SamplingFreq	Sampling frequency of the on-board clock.
bOnBoardClk	True to use the on-board clock otherwise false. False is the default value.

Return value: Sampling frequency of the on-board clock.

Notes: This function configures the clock of the SMT381.

SetTrigger

Prototype:
```
void SetTrigger(bool bStart);
```

Parameters:

bStart	Set the trigger to 0 of 1. The trigger is active high.

Return value: None.

Notes: This function sets the trigger of the SMT381.

Reset

Prototype:
```
void Reset(bool bSDB, bool bDAC, bool bDCM);
```

Parameters:

bSDB	SDB reset. 0 = inactive, 1 = active.
bDAC	DAC reset, active on falling edge. Reset all FPGA internal registers and DAC.
bDCM	DCM reset. 0 = inactive, 1 = active.

Return value: None.

Notes: This function resets the SMT381.

The SMT381 enumerated types

Enumerated type	Description
enum **DATAMODE** { direct, memory };	Selects the data mode: direct = the input data comes from the FPGA, memory = the input data comes from the DAC memory.

3.3.5 SMT384 interface

The SMT384 interface can be used with a SMT384 plugged in a Sundance carrier board.

This section describes:

- the SMT384 functions
  - ResetDevices
  - ConfigureADCReg
  - ConfigureADCClock
  - Acquire
- the SMT384 structures
  - CLKMODE384

The SMT384 functions

All the following functions can throw an SMTExc exception.

ResetDevices

Prototype: `void ResetDevices(void);`

Parameters: None.

Return value: None.

Notes: This function does a global reset of the SMT384 (ADCs, Clock…).

ConfigureADCReg

Prototype: `void ConfigureADCReg(OUTPUT output, nVCXOTYPE vcxotype, unsigned int nClkDiv);`

	output	Type of data the ADCs output. (See enum OUTPUT).
Parameters:	nClkDiv	VCO clock divider to set the sampling frequency.
	vcxotype	Select the right VCXO used on the SMT384: 100MHz or 245.76 MHz. (See enum VCXOTYPE).

Return value: None.

Notes: This function configures the ADC registers of the SMT384.

ConfigureADCClock

Prototype: `void ConfigureADCClk(CLKMODE384 clkmode);`

Parameters:	clkmode	ADC clock mode parameters. (See struct CLKMODE384).

Return value: None.

Notes: This function configures the clock for the ADCs on the SMT384.

Acquire

Prototype: `void Acquire(CLKMODE384 clkmode, unsigned int nBytes);`

Parameters:	clkmode	ADC clock mode parameters. (See struct CLKMODE384).
	nBytes	Number of bytes to acquire.

Return value: None.

Notes: This function set the ADCs on the SMT384 to acquire data.

The SMT384 structures

struct CLKMODE384	
`bool bExtRef`	true = external reference; false = on-board 10 MHz reference. The default value is false.
`bool bExtClk`	true = external clock; false = on-board VCXO.

	The default value is false.
bool bABTrigInvert	true = trigger polarity channel A and B; false = Non-inverting. The default value is false.
bool bABTrigExt	true = trigger selection channel A and B; false = internal trigger. The default value is false.
bool bCDTrigInvert	true = trigger polarity channel C and D; false = Non-inverting. The default value is false.
bool bCDTrigExt	true = trigger selection channel C and D; false = internal trigger. The default value is false.
CHSEL ABChSel	Channel A and B selection. The default value is channelab.
CHSEL CDChSel	Channel C and D selection. The default value is channelab.
bool bAB2sComplement	True = ADC A and B output 2's complement samples False = ADC A and B output binary samples. The default value is false.
bool bCD2sComplement	True = ADC C and D output 2's complement samples False = ADC C and D output binary samples. The default value is false.

3.3.6 SMT391 interface

The SMT391 interface can be used with a SMT391 plugged in a Sundance carrier board.

This section describes:

- the SMT391 functions
  - Reset
  - ConfigureADCReg
  - ConfigureADCClock
  - Acquire
  - ADCReset
  - SetADCMode
  - SetClkMode
  - GetFirmwareVersion
  - GetTemp
  - GetSN
- the SMT391 structures
  - SETADC391

- the SMT391 enumerated types
  - CLKMODE391

The SMT391 functions

All the following functions can throw an SMTExc exception.

Reset

Prototype: `void Reset(bool bGlobal, bool bADC, bool bUser);`

Parameters:

bGlobal True to do a global reset otherwise false.

bADC True to do an ADC reset otherwise false.

bUser True to do a user reset otherwise false.

Return value: None.

Notes: This function reset the SMT391.

ConfigureADCReg

Prototype: `void ConfigureADCReg(unsigned int &nSamplingFreq, SETADC391 setadc);`

Parameters:

nSamplingFreq Sampling frequency of the on-board clock.

setadc ADC parameters. (See struct SETADC391).

Return value: None.

Notes: This function configures the ADC registers of the SMT391.

ConfigureADCClock

Prototype: `unsigned int ConfigureADCClock(unsigned int nRf_SamplingFreq, bool bOnBoardClk);`

Parameters:

nRf_SamplingFreq Sampling frequency of the on-board clock.

bOnBoardClk True to use the on-board clock otherwise false. False is the default value.

Return value: Sampling frequency of the on-board clock.

Notes: This function configures the clock of the SMT391.

Acquire

Prototype: `void Acquire(unsigned int nBytes=0, bool bDDR=false);`

Parameters:

nBytes	Number of bytes to acquire. If no DDR memory has been implemented in the FPGA firmware, the acquisition is continue and nBytes is not used. 0 is the default value.
bDDR	True when DDR memory is implemented in the FPGA firmware otherwise false. False is the default value.

Return value: None.

Notes: This function configures the SMT391 to start acquiring data.

StopAcquire

Prototype: `void StopAcquire(HOST_LINK hostlink);`

Parameters:

hostlink	Communication link on which to write the data. Host comport per default. (See enum HOST_LINK).

Return value: None.

Notes: This function stops the ADC to acquire data.

ADCReset

Prototype: `void ADCReset(void);`

Parameters: None.

Return value: None.

Notes: This function resets the ADCs.

SetADCMode

Prototype: `void SetADCMode(unsigned int n);`

Parameters:

n	ADC mode value.

Return value: None.

Notes: This function sets the ADC mode.

SetClkMode

Prototype: `void SetClkMode(CLKMODE391 clkmode);`

Parameters: clkmode ADC clock mode parameters. (See struct CLKSMT391).

Return value: None.

Notes: This function sets the clock mode.

GetFirmwareVersion

Prototype: `unsigned int GetFirmwareVersion(void);`

Parameters: None.

Return value: None.

Notes: This function gets the firmware version. Before using this function the user has to make sure this functionality is part of the firmware used to program the FPGA.

GetTemp

Prototype: `void GetTemp(TEMP *pTemp);`

Parameters: pTemp Structure that receives the modules' temperatures (See struct TEMP).

Return value: None.

Notes: This function gets the temperatures on different part of the hardware. Before using this function the user has to make sure this functionality is part of the firmware used to program the FPGA.

GetSN

Prototype: `void GetSN(SN *pSN);`

Parameters: pSN Structure that receives the modules' temperatures. (See struct SN).

Return value: None.

Notes: This function gets the FPGA module serial number. Before using this function the user has to make sure this functionality is part of the firmware used to program the FPGA.

The SMT391 structures

struct SETADC391	
bool bItoQ	True = The input for the channels I and Q is the same. False = Channels I and Q have different inputs. The default value is true.
unsigned int nAnalogGain	Analog gain. The default value is 128.
unsigned int nOffsetComp	Offset compensation. The default value is 0.
bool bTest	True = the ADCs outputs test data. False = the ADCs outputs data from theirs inputs. The default value is false.
unsigned int nDrda	Set the value of the DRDA between 0 and 7. The default value is 7.

The SMT391 enumerated types

Enumerated type	**Description**

enum **CLKMODE391** {
clk_mode_I_I_Q_Q,
clk_mode_Q_I_Q_Q,
clk_mode_I_I_I_Q,
clk_mode_Q_I_I_Q
};
	Selects the clock mode.

3.3.7 SMT941 interface

The SMT941 interface can be used with a SMT941 plugged in a Sundance carrier board.

This section describes:

- the SMT941 functions
  - ResetDevices
  - ConfigureADCReg
  - AcquireAB
  - AcquireCD
  - GetFirmwareVersion
  - GetError
- the SMT941 structures
  - CLKMODE941
- the SMT941 enumerated types
  - ERRORS
  - CLOCK_DIVIDER

The SMT941 functions

All the following functions can throw an SMTExc exception.

ResetDevices

Prototype: `void ResetDevices(void);`

Parameters: None.

Return value: None.

Notes: This function reset the SMT941.

ConfigureADCReg

Prototype: `int ConfigureADCReg(OUTPUT output, CLKMODE941 clkmode);`

Parameters:

output Type of data the ADCs output. (See enum OUTPUT).

clkmode Select the clock parameters. (See struct CLKMODE941).

Return value: None.

Notes: This function configures the ADC registers of the SMT941.

AcquireAB

Prototype: `int AcquireAB(unsigned int nBytes, unsigned chart *a, unsigned short *b);`

	nBytes	Number of bytes to acquire.
Parameters:	a	Data sampled from channel A.
	b	Data sampled from channel B.
Return value:	Return 0 if successful otherwise 1.	
Notes:	This function acquires data on channel A and channel B of the SMT941.	

AcquireCD

Prototype:
```
int AcquireCD(unsigned int nBytes, unsigned chart *c, unsigned short *d);
```

	nBytes	Number of bytes to acquire.
Parameters:	c	Data sampled from channel C.
	d	Data sampled from channel D.
Return value:	Return 0 if successful otherwise 1.	
Notes:	This function acquires data on channel C and channel D of the SMT941.	

GetFirmwareVersion

Prototype:
```
unsigned int GetFirmwareVersion(void);
```

Parameters:	None.
Return value:	None.
Notes:	This function gets the firmware version. Before using this function the user has to make sure this functionality is part of the firmware used to program the FPGA.

GetError

Prototype:
```
char * GetError(int err);
```

Parameters:	err	Error code.
Return value:	The description of the error code.	
Notes:	This function returns the description of the error code enter as parameter.	

The SMT941 structures

struct CLKMODE941	
bool bExtRef	True = External reference. False = On-board 10 MHz reference. The default value is false.
bool bExtClk	True = External Clock. False = On-board VCXO. The default value is false.
bool bTrigInvert	Trigger polarity True = Inverting. False = Non-Inverting. The default value is false.
bool bTrigExt	Trigger selection True = External trigger. False = Internal trigger. The default value is false.
bool b2sComplement	True = External reference. False = On-board 10 MHz reference. The default value is false.
unsigned int nClkDiv	Set the value of the clock divider. The default value is div_by_one.

The SMT941 enumerated types

Enumerated type	**Description**

enum **ERRORS** {
ERR_SUCCESS = 0,
ERR_CLK_REG = 1,
ERR_CLK_LOCKED = 2,
ERR_ADCAB_DCM_LOCKED = 3,
ERR_ADCCD_DCM_LOCKED = 4
};
	Error codes.

enum **CLOCK_DIVIDER** {	Selects the clock divider.

```
  div_by_one   = 0,
  div_by_two   = 1,
  div_by_three = 2,
  div_by_four  = 3,
  div_by_five  = 4
};
```

4 Examples implementation

The SMT1036 provides several examples to illustrate the used of the Sundance's FPGA library.

There is an example for each supported FPGA module and a more generic one.

Those examples are organised in three folders:

- Firmware
- Host
- Output

4.1 Firmware

This folder contains several firmware for the FPGA-only module.

The bitstream have been compiled with 3L Diamond FPGA and therefore are *.app files. The "*readme.txt*" file contains some information about the firmware. The source code can be provided on demand but an NDA (Non Disclosure Agreement) has to be signed first.

Remark: The FPGA library works with the basic functionalities implemented in the firmware supplied with the SMT1036. If bitstreams other than the ones provided with the SMT1036 are used with the FPGA library, the software will not work properly anymore. In those circumstances Sundance will decline all responsibilities.

4.2 Host

This folder contains the source code for the host.

The source code is similar for all the supported FPGA-only modules:

```
1- Open the FPGA library.
2- Reset the board and TIM.
3- Configure the FPGA.
4- Configure the ADCs/DACs.
5- Start the acquisition on the ADCs or trigger the DACs.
6- Save data to file (ADCs).
```

4.3 Output

This folder contains the executable application.

This executable is a very simple console application aimed to show the basic use of the SMT1036 libraries. But more complicated and flexible user interfaces can be developed using the functionalities of the SMT1036 libraries.