

# FX2 USB Light IP

For Saturn SX1

User Manual

Product Info	
Product Manager	Marc Oberholzer
Author(s)	Marc Oberholzer (MO)
Reviewer(s)	Martin Heimlicher (MH)
Version	1.00
Date	14.12.2009

## Copyright reminder

Copyright © 2009 by Enclustra GmbH, Switzerland. All rights are reserved.

Unauthorized duplication of this document, in whole or in part, by any means is prohibited without the prior written permission of Enclustra GmbH, Switzerland.

Although Enclustra GmbH believes that the information included in this publication is correct as of the date of publication, Enclustra GmbH reserves the right to make changes at any time without notice.

All information in this document is strictly confidential and may only be published by Enclustra GmbH, Switzerland.

All referenced trademarks are the property of their respective owners.

## Document History

Version	Date	Author	Comment
1.00	14.12.2009	MO	<ul style="list-style-type: none"><li>Adapted to the current IP core and reference design revisions.</li></ul>
0.02	15.06.2009	MO	<ul style="list-style-type: none"><li>Updated directory structure (Figure 2, page 6)</li><li>Updated section 4.2 Host PC Software</li></ul>
0.01	14.05.2009	MO	<ul style="list-style-type: none"><li>Preliminary</li></ul>

## Applicable Revisions

Item	Revision
FX2 USB Light IP Core Revision	1.00
Reference Design FPGA FW Revision	1.00
Reference Design DMA Demo SW Revision	1.00
Reference Design Read Demo SW Revision	1.00
Reference Design Module Status Demo SW Revision	1.00

# Table of Contents

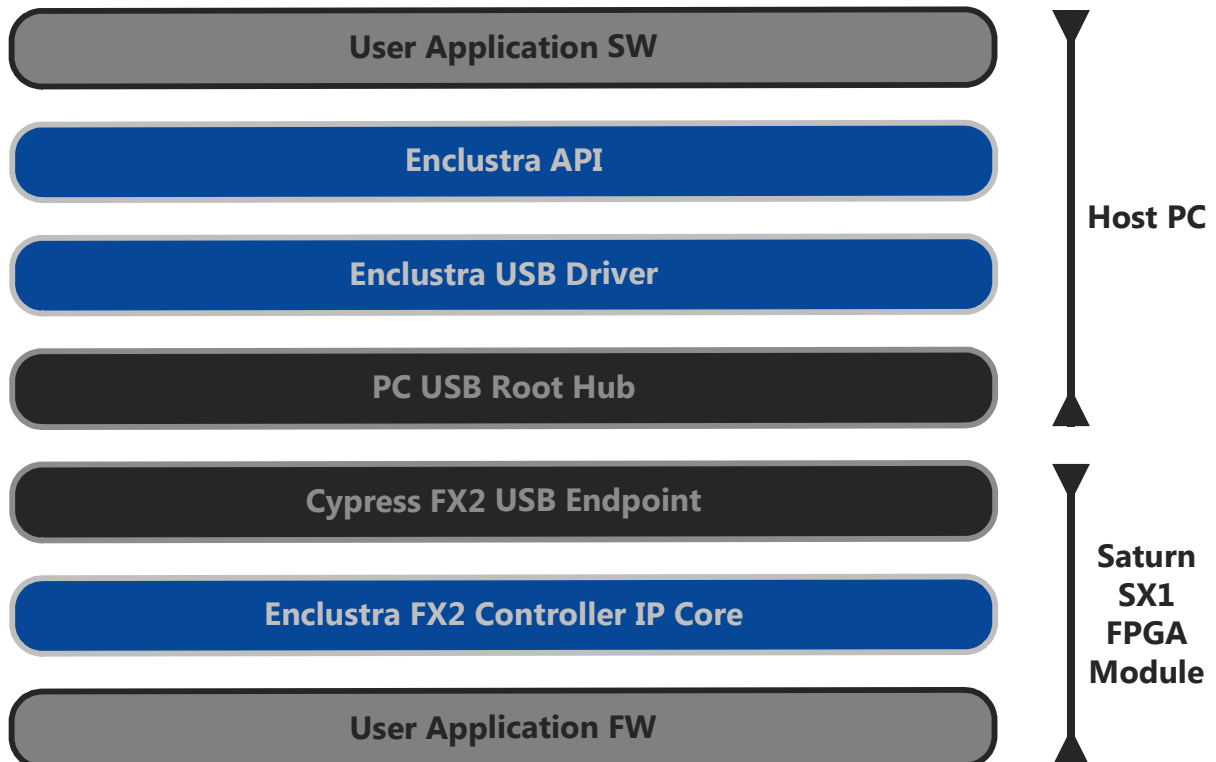
<b>1</b>	<b>Overview.....</b>	<b>5</b>
<b>1.1</b>	<b>Funcional Description.....</b>	<b>5</b>
<b>1.2</b>	<b>Deliverables.....</b>	<b>6</b>
1.2.1	Windows.....	6
1.2.2	Linux.....	6
<b>1.3</b>	<b>Installation.....</b>	<b>6</b>
1.3.1	Windows.....	6
1.3.2	Linux.....	9
<b>2</b>	<b>IP Core .....</b>	<b>10</b>
<b>2.1</b>	<b>Overview .....</b>	<b>10</b>
<b>2.2</b>	<b>Interfaces .....</b>	<b>10</b>
2.2.1	User Clock and Reset Interface .....	10
2.2.2	FX2 Clock and Reset Interface .....	11
2.2.3	External FX2 Interface (synchronous to Fx2_IfClk) .....	11
2.2.4	ClustraBus Interface (synchronous to Clk) .....	12
2.2.5	DMA Interface (synchronous to Clk) .....	13
2.2.6	IP Revision Interface .....	13
<b>2.3</b>	<b>Files.....</b>	<b>14</b>
<b>2.4</b>	<b>Resource Usage .....</b>	<b>14</b>
<b>2.5</b>	<b>Development Tools.....</b>	<b>14</b>
<b>3</b>	<b>Application Programming Interface (API) .....</b>	<b>15</b>
<b>3.1</b>	<b>C / C++ .....</b>	<b>15</b>
3.1.1	Files.....	15
3.1.2	Scalar Types .....	15
3.1.3	Enumerated Types.....	15
3.1.4	Data Structures .....	18
3.1.5	Functions .....	19
<b>3.2</b>	<b>Java.....</b>	<b>26</b>
<b>3.3</b>	<b>.NET .....</b>	<b>26</b>
<b>3.4</b>	<b>MATLAB.....</b>	<b>27</b>
<b>3.5</b>	<b>LabView.....</b>	<b>27</b>
<b>4</b>	<b>Reference Design.....</b>	<b>28</b>
<b>4.1</b>	<b>FPGA Firmware .....</b>	<b>28</b>
4.1.1	Functional Description.....	28
4.1.2	Design Hierarchy.....	28

4.1.3	Interfaces.....	29
4.1.4	Memory Map.....	29
4.1.5	Register description.....	30
4.1.6	LED Activity Interpretation .....	31
4.1.7	Files.....	31
4.1.8	Development Tools.....	32
<b>4.2</b>	<b>Host PC Software .....</b>	<b>32</b>
4.2.1	C / C++ .....	32
4.2.2	Java .....	33
4.2.3	.NET .....	33
4.2.4	MATLAB .....	33
4.2.5	LabView.....	33
<b>5</b>	<b>Use Cases .....</b>	<b>34</b>
<b>5.1</b>	<b>Acquiring and Processing Data from a Streaming Data Source .....</b>	<b>34</b>
5.1.1	Overview.....	34
5.1.2	FPGA Firmware.....	35
5.1.3	Host PC Software .....	36
<b>6</b>	<b>Conventions for Register Descriptions .....</b>	<b>37</b>
<b>6.1</b>	<b>Register Field Types .....</b>	<b>37</b>
<b>6.2</b>	<b>Reset Values .....</b>	<b>37</b>
<b>6.3</b>	<b>Identifiers .....</b>	<b>38</b>
<b>6.4</b>	<b>Size and Address Information .....</b>	<b>38</b>
<b>6.5</b>	<b>Number Formats .....</b>	<b>38</b>
6.5.1	Fixed Point.....	38
6.5.2	Floating Point .....	38

# 1 Overview

## 1.1 Funcional Description

The FX2 USB Light IP core, together with the software drivers and the API, allows connecting one or more Enclustra Saturn SX1 FPGA module(s) to a host PC via an USB 2.0 high speed interface. Figure 1 shows the simplified system layering of a Saturn SX1 FPGA module which is connected to a host PC.



**Figure 1: Simplified system layering**

The Enclustra FX2 USB Light IP core accepts and processes read and write requests received from the FX2 USB device located on the Saturn SX1 FPGA module. It also supports DMA data transfers from the FPGA to the host PC initiated by the user design implemented in the FPGA.

The Enclustra FX2 USB driver enables a Saturn SX1 FPGA module to be recognized and accessed by a host PC running Windows or Linux.

The Enclustra FX2 USB API allows to develop user application software which communicates to the Saturn SX1 FPGA module via an USB 2.0 high-speed interface.

## 1.2 Deliverables

### 1.2.1 Windows

The FX2 USB Light IP comes as an automated installer file (.exe). It generates the directory structure shown in Figure 2 within the C:\Program Files\ folder.<sup>1</sup>

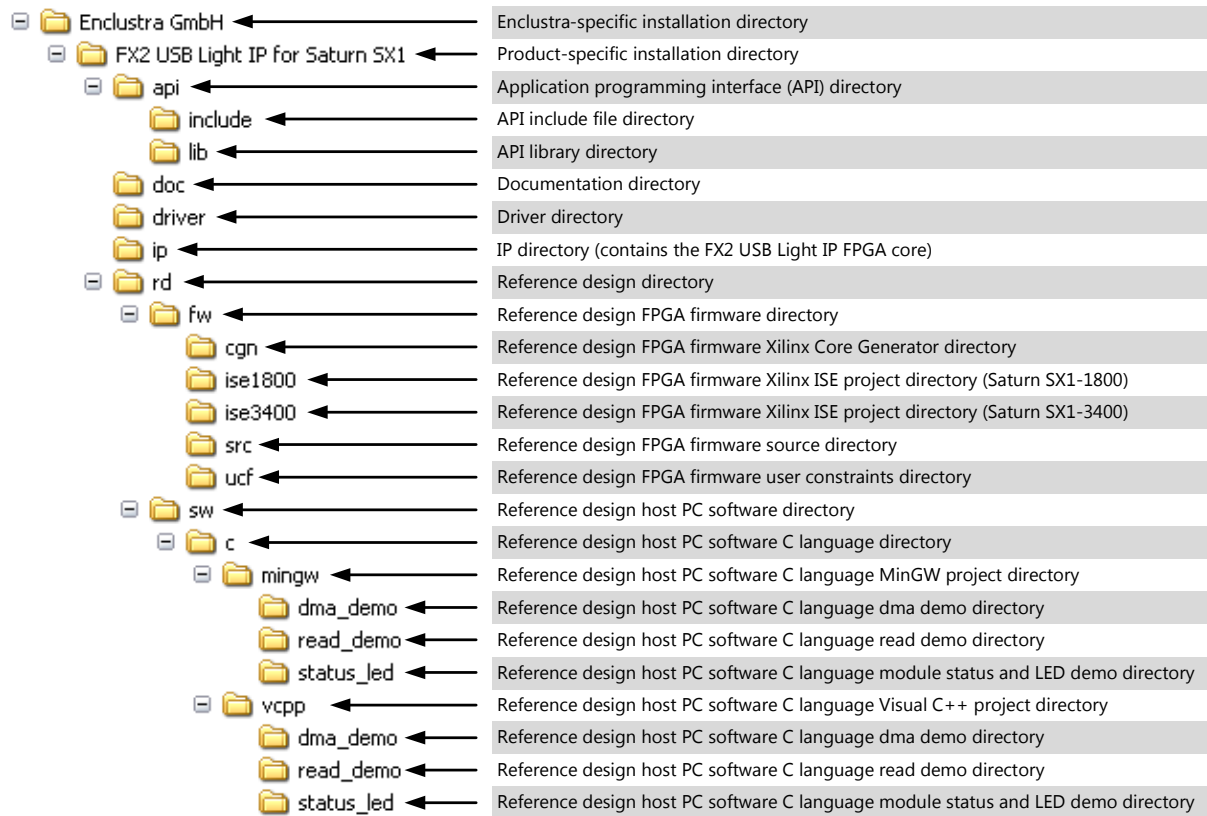


Figure 2: Installation directory structure

### 1.2.2 Linux

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## 1.3 Installation

### 1.3.1 Windows

#### 1.3.1.1 General

1. Make sure that **no** Saturn SX1 FPGA module is connected to the host PC.

<sup>1</sup> This is the default installation path. An alternative installation path may be chosen during installation.

2. Double-click the automated installer file (Fx2UsbLightIp\_Installer\_v1.00.exe) and follow the on-screen instructions.

### 1.3.1.2 Drivers

1. Make sure to uninstall **any** older version of the Saturn SX1 USB drivers. The Saturn SX1 USB drivers are listed in the device manager in the category "FPGA Module" as "EncFX2".
2. Connect a properly set up and powered Saturn Starter / Saturn SX1 combination to a USB 2.0 port of the host PC. Make sure to use the "USB" (and not the "UART") port of the Saturn Starter base board.
3. Follow the "Found New Hardware Wizard" instructions like shown in Figure 3 to Figure 6.
4. Select the directory "Enclustra GmbH\FX2 USB Light IP for Saturn SX1\driver" in the dialog shown in screenshot 3 (Figure 5).



Figure 3: Windows driver installation screenshot 1



Figure 4: Windwos driver installation screenshot 2

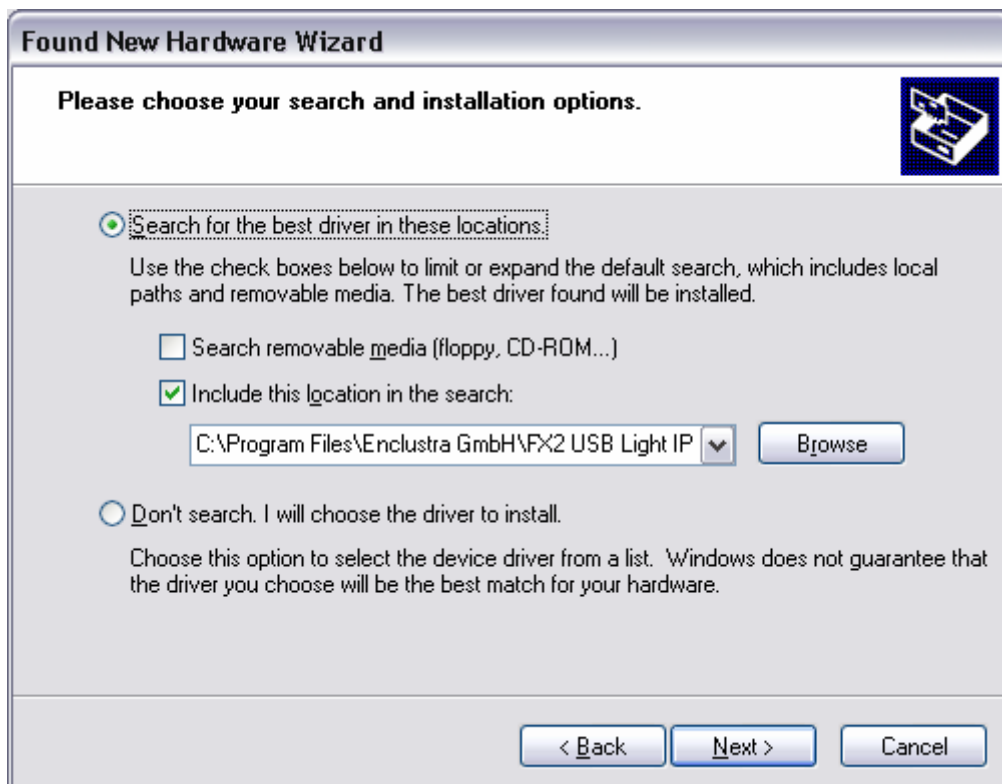


Figure 5: Windwos driver installation screenshot 3



Figure 6: Windwos driver installation screenshot 4

### 1.3.2 Linux

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

# 2 IP Core

## 2.1 Overview

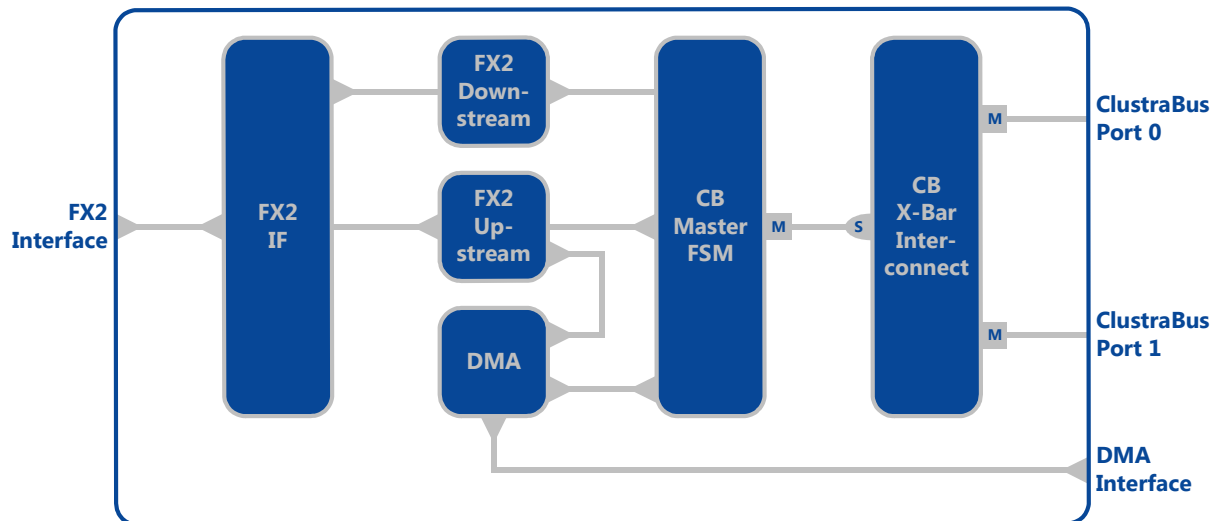


Figure 7: IP core block diagram

The Enclustra FX2 USB Light IP core mainly consists of an FX2-related part and a ClustraBus-related part.

The FX2-related part handles the FX2 interface protocol as well as up- and downstream processing. It is clocked by the FX2 interface clock which has a fixed frequency of 48 MHz.

The ClustraBus-related part provides two memory-mapped user ports which act as ClustraBus masters. In addition, a DMA interface is provided, allowing the FPGA user design to initiate data transfers from the FPGA to the host PC. The ClustraBus-related part may be clocked by a user-generated clock.

## 2.2 Interfaces

### 2.2.1 User Clock and Reset Interface

This interface is used to provide user clock and reset signals to the IP core. The ClustraBus and DMA interfaces are synchronous to the user clock. Table 1 shows the interface signals.

Signal Name	Width	Direction	Description
Clk	1	In	User clock input. Frequency must be between 1 MHz and 250 MHz.
Rst	1	In	User reset input. Must be synchronized to the user clock (Clk).

Table 1: IP core user clock and reset interface description

## 2.2.2 FX2 Clock and Reset Interface

This interface provides the FX2 interface clock to the user. This clock may be used to drive the user design<sup>2</sup>. Table 2 shows the interface signals.

Signal Name	Width	Direction	Description
Clk_Fx2	1	Out	A copy of the 48 MHz external FX2 interface clock (Fx2_IfClk). This clock may be used to drive the user design. <sup>2</sup>
Rst_Fx2	1	Out	A copy of the user reset input (Rst), synchronized to Clk_Fx2.

Table 2: IP core FX2 clock and reset interface description

## 2.2.3 External FX2 Interface (synchronous to Fx2\_IfClk)

This interface connects the FX2 USB Light IP core to the external FX2 device. Pin assignments, I/O standards and timing constraints are provided within the user constraints file (see Table 8 on page 14). Table 3 shows the interface signals.

Signal Name	Width	Direction	Description
Fx2_IfClk	1	In	48 MHz interface clock received from the external FX2 device.
Fx2_FifoAddr	2	Out	FIFO address to the external FX2 device
Fx2_SICs_N	1	Out	Chip select to the external FX2 device (active low)
Fx2_SIW_r_N	1	Out	Write enable to the external FX2 device (active low)
Fx2_SIRd_N	1	Out	Read enable to the external FX2 device (active low)
Fx2_SIOe_N	1	Out	Output enable to the external FX2 device (active low)
Fx2_PktEnd_N	1	Out	Packet end indicator to the external FX2 device (active low)
Fx2_Flag	3	In	Flags from the external FX2 device
Fx2_Fd	16	Inout	FIFO data to/from the external FX2 device
Fx2_Int1_N	1	Out	Interrupt to the external FX2 device (active low, not used, set to '1' constantly)

Table 3: IP core external FX2 interface description

---

<sup>2</sup> In this case, the FX2 clock output (Clk\_FX2) has to be connected to the user clock input (Clk) for the IP core to work properly.

## 2.2.4 ClustraBus Interface (synchronous to Clk)

### 2.2.4.1 Signals

This interface allows connecting two ClustraBus slaves to the FX2 USB Light IP. If more slaves need to be connected to the FX2 USB light IP, a ClustraBus interconnect<sup>1</sup> is required. Table 4 shows the interface signals.

Signal Name	Width	Direction	Description
M_CmdReq	2	Out	Command request and slave select signal (to 2 slaves)
M_CmdLock	2	Out	Lock signal (to 2 slaves)
M_CmdAck	2	In	Command acknowledge signal (from 2 slaves)
M_CmdWrite	2	Out	Command write signal (to 2 slaves)
M_CmdAddr	2*30	Out	Command address vector (to 2 slaves, )
M_DataVld	2	Out	Data valid signal (to 2 slaves)
M_DataLast	2	Out	Data last signal (to 2 slaves)
M_DataRdy	2	In	Data ready signal (from 2 slaves)
M_WrData	2*32	Out	Write data vector (to 2 slaves)
M_WrByteEna	2*4	Out	Write byte enable vector (to 2 slaves)
M_RdData	2*32	In	Read data vector (from 2 slaves)
M_RdVld	2	In	Read data valid signal (from 2 slaves)
M_RdLast	2	In	Read data last signal (from 2 slaves)

**Table 4: IP core ClustraBus interface description**

### 2.2.4.2 Protocol

The ClustraBus protocol is described in detail in the ClustraBus Protocol Specification available at the Enclustra website.<sup>2</sup>

### 2.2.4.3 Memory Map

The 4 GB total memory space is divided into two equally sized 2GB sections. Table 5 shows the respective address ranges.

**Attention:** Only dword (4 bytes) accesses are supported by the FX2 USB Light IP core. This means that transfer start byte addresses and transfer byte counts must always be aligned to four bytes.

ClustraBus Port	Byte Address Range	Dword Address Range
0	0x00000000 .. 0x7FFFFFFC	0x00000000 .. 0x1FFFFFFF
1	0x80000000 .. 0xFFFFFFFF	0x20000000 .. 0x3FFFFFFF

**Table 5: IP core ClustraBus port address ranges**

## 2.2.5 DMA Interface (synchronous to Clk)

### 2.2.5.1 Signals

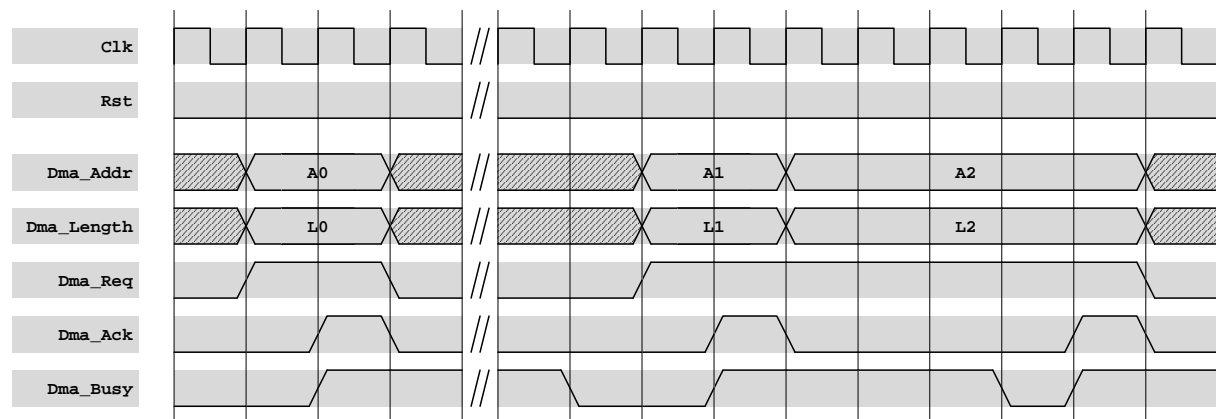
This interface allows the FPGA user design to initiate data transfers from the FPGA to the host PC's receive buffer. Table 6 shows the interface signals.

Signal Name	Width	Direction	Description
Dma_Addr	30	In	DMA transfer source address. This is a dword (4 bytes) address!
Dma_Length	30	In	DMA transfer length in dwords (4 bytes)
Dma_Req	1	In	DMA request signal
Dma_Ack	1	Out	DMA acknowledge signal

**Table 6: IP core DMA interface description**

### 2.2.5.2 Protocol

The DMA interface protocol makes use of a common handshake (request/acknowledge) mechanism. Figure 8 shows some example waveforms.



**Figure 8: IP core DMA interface waveforms**

The user design applies valid address and length values and raises the request signal. The FX2 USB Light IP core acknowledges the acceptance of the address and length values with a pulse on the acknowledge signal. The busy signal indicates that a DMA transfer is currently in process. The FX2 USB Light IP core does not accept new DMA transfer requests while the busy signal is active.

**Attention:** While *Dma\_Req* is high, valid address and length values must be applied to *Dma\_Addr* and *Dma\_Length*. These values may only be changed if a pulse on the *Dma\_Ack* signal is seen.

## 2.2.6 IP Revision Interface

This interface allows the user design to read the actual IP revision. As the IP revision is an IP core internal constant, no clock synchronicity requirements do exist. Table 7 shows the interface signals.

Signal Name	Width	Direction	Description
Ip_Revision	16	Out	Bits 15:8 contain the major revision number (0..255), bits 7:0 contain the minor revision number (0..99)

**Table 7: IP core IP revision interface description**

## 2.3 Files

File Name	Description
fx2_usb_light.vhd	This is a wrapper file containing an instance of the FX2 USB Light IP netlist, as well as some I/O and clock handling. This is the file which has to be instantiated in the user design.
fx2_usb_light_nl.ngc	This file contains the FX2 USB Light IP netlist. The path of this file has to be added to the search directory list of the user design Xilinx ISE project.
fx2_usb_light.ucf	This file contains the Xilinx ISE user constraints (pin assignments, I/O standards and timing constraints) for the FX2 USB Light IP. This file has to be added to the Xilinx ISE project of the user design.

**Table 8: IP core files**

## 2.4 Resource Usage

FPGA Type	FFs	LUTs	BRAMs	IO Pins
XC3SD3400A -5CSG484C	911	1018	3-5 <sup>3</sup>	30

**Table 9: IP core resource usage**

## 2.5 Development Tools

The FX2 USB Light IP netlist file has been generated with Xilinx ISE 11.3.

---

<sup>3</sup> The number of used BRAMs is dependent on the general resource usage of the user design as well as on the Xilinx ISE project settings.

# 3 Application Programming Interface (API)

## 3.1 C / C++

### 3.1.1 Files

File Name	Directory	Description
EncFX2.h	.\api\include	API header file
libEncFX2.dll	.\api\lib\linux	API Linux shared object file <sup>4</sup>
libEncFX2.dll	.\api\lib\windows	API WIN32 DLL file

Table 10: C/C++ API files

### 3.1.2 Scalar Types

#### 3.1.2.1 ENCFX2\_HANDLE

This type is used to represent a handle to a Saturn SX1 FPGA module.

```
typedef void* ENCFX2_HANDLE;
```

Code Snippet 1: ENCFX2\_HANDLE scalar type (C/C++ API)

### 3.1.3 Enumerated Types

**Attention:** More enumerated type fields than documented in this user manual may actually exist in the C/C++ header file. Undocumented enumerated type fields are for internal use only and should not be used in the user application software.

#### 3.1.3.1 ENCFX2\_STATUS

This type is used to represent the API function return values.

---

<sup>4</sup> Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

```

typedef enum{
    ENCFX2_SUCCESS,
    ENCFX2_MAX_INFO,
    ENCFX2_WARNING,
    ENCFX2_MAX_WARNING,
    ENCFX2_ERROR,
    ENCFX2_INTERNAL,
    ENCFX2_MEMORY,
    ENCFX2_NULLPTR,
    ENCFX2_TIMEOUT,
    ENCFX2_FILE_NOTFOUND,
    ENCFX2_FILE_ERROR,
    ENCFX2_ILLEGAL_CARD,
    ENCFX2_PROGRAM_FAILED,
    ENCFX2_ILLEGAL_HANDLE,
    ENCFX2_ILLEGAL_BITSTREAM,
    ENCFX2_INVALID_PARTTYPE,
    ENCFX2_ILLEGAL_STATUS,
    ENCFX2_WRONG_STATE,
    ENCFX2_UNKNOWN,
    ENCFX2_MAX_ERROR,
} ENCFX2_STATUS;

```

**Code Snippet 2: ENCFX2\_STATUS enumerated type (C/C++ API)**

Field	Description
ENCFX2_SUCCESS	The operation was successful
<i>ENCFX2_MAX_INFO</i>	<i>For internal use only</i>
<i>ENCFX2_WARNING</i>	<i>For internal use only</i>
<i>ENCFX2_MAX_WARNING</i>	<i>For internal use only</i>
<i>ENCFX2_ERROR</i>	<i>For internal use only</i>
ENCFX2_INTERNAL	An unspecified internal error occurred
ENCFX2_MEMORY	Not enough memory
ENCFX2_NULLPTR	A null pointer is passed where it is not allowed
ENCFX2_TIMEOUT	Operation timed out
ENCFX2_FILE_NOTFOUND	The specified file could not be found
ENCFX2_FILE_ERROR	There was an error handling the specified file
ENCFX2_ILLEGAL_CARD	The specified card does not exist or is not connected
ENCFX2_PROGRAM_FAILED	Programming of the FPGA failed
ENCFX2_ILLEGAL_HANDLE	The given handle is invalid
ENCFX2_ILLEGAL_BITSTREAM	The given FPGA bitstream is malformed
ENCFX2_INVALID_PARTTYPE	The given FPGA bitstream is not intended for the FPGA type
ENCFX2_ILLEGAL_STATUS	Status code does not exist

ENCFX2_WRONG_STATE	The message state machine in the wrong state to perform the desired function
ENCFX2_UNKNOWN	An unknown value is passed
ENCFX2_MAX_ERROR	<i>For internal use only</i>

**Table 11: ENCFX2\_STATUS enumerated type (C/C++ API)**

### 3.1.3.2 ENCFX2\_FPGATYPE

This type is used to represent the FPGA type equipped on the Saturn SX1 FPGA module.

```
typedef enum
{
    ENCFX2_FPGA_UNKNOWN,
    ENCFX2_FPGA_XC3SD1800,
    ENCFX2_FPGA_XC3SD3400,
} ENCFX2_FPGATYPE;
```

**Code Snippet 3: ENCFX2\_FPGATYPE enumerated type (C/C++ API)**

Field	Description
ENCFX2_FPGA_UNKNOWN	The FPGA type is unknown
ENCFX2_FPGA_XC3SD1800	The FPGA type is XC3SD1800A-4CSG484C
ENCFX2_FPGA_XC3SD3400	The FPGA type is XC3SD3400A-5CSG484C

**Table 12: ENCFX2\_FPGATYPE enumerated type (C/C++ API)**

### 3.1.3.3 ENCFX2\_CARDSTATE

This type is used to select which parameter should be retrieved with a call to either EncFX2GetCardstateFlag() or EncFX2GetCardstateValue().

```
typedef enum {
    ENCFX2_CARDSTATE_NONE,
    ENCFX2_CARDSTATE_1V2,
    ENCFX2_CARDSTATE_1V8,
    ENCFX2_CARDSTATE_2V5,
    ENCFX2_CARDSTATE_3V3,
    ENCFX2_CARDSTATE_5V0,
    ENCFX2_CARDSTATE_TEMP,
    ENCFX2_CARDSTATE_MAX,
} ENCFX2_CARDSTATE;
```

**Code Snippet 4: ENCFX2\_CARDSTATE enumerated type (C/C++ API)**

Field	Description
ENCFX2_CARDSTATE_NONE	<i>For internal use only</i>
ENCFX2_CARDSTATE_1V2	The 1.2 V supply voltage parameter value/flag shall be retrieved
ENCFX2_CARDSTATE_1V8	The 1.8 V supply voltage parameter value/flag shall be retrieved
ENCFX2_CARDSTATE_2V5	The 2.5 V supply voltage parameter value/flag shall be retrieved
ENCFX2_CARDSTATE_3V3	The 3.3 V supply voltage parameter value/flag shall be retrieved

ENCFX2_CARDSTATE_5V0	The 5.0 V supply voltage parameter value/flag shall be retrieved
ENCFX2_CARDSTATE_TEMP	The temperature parameter value/flag shall be retrieved
ENCFX2_CARDSTATE_MAX	<i>For internal use only</i>

**Table 13: ENCFX2\_CARDSTATE enumerated type (C/C++ API)**

### 3.1.3.4 ENCFX2\_CARDSTATE\_FLAG

This type is used to represent the status of a parameter requested by a call to EncFX2GetCardstateFlag().

```
typedef enum {
    ENCFX2_CARDSTATE_OK,
    ENCFX2_CARDSTATE_RESERVED,
    ENCFX2_CARDSTATE_WARNING,
    ENCFX2_CARDSTATE_ERROR,
} ENCFX2_CARDSTATE_FLAG;
```

**Code Snippet 5: ENCFX2\_CARDSTATE\_FLAG enumerated type (C/C++ API)**

Field	Description
ENCFX2_CARDSTATE_OK	The requested parameter value is in normal range
ENCFX2_CARDSTATE_RESERVED	<i>For internal use only</i>
ENCFX2_CARDSTATE_WARNING	The requested parameter value is between normal and error range
ENCFX2_CARDSTATE_ERROR	The requested parameter value is in error range

**Table 14: ENCFX2\_CARDSTATE\_FLAG enumerated type (C/C++ API)**

## 3.1.4 Data Structures

### 3.1.4.1 ENCFX2\_CARDINFO

This data structure is used to hold the general configuration information of a Saturn SX1 FPGA module.

```
typedef struct {
    unsigned int    ProductNumber;
    unsigned int    SerialNumber;
    unsigned char   FirmwareVersionMaj;
    unsigned char   FirmwareVersionMin;
    unsigned char   PicVersionMaj;
    unsigned char   PicVersionMin;
    ENCFX2_FPGATYPE FPGAType;
    unsigned char   DDR2Size;
    unsigned char   SSRAMSize;
    unsigned char   FlashSize;
} ENCFX2_CARDINFO;
```

**Code Snippet 6: ENCFX2\_CARDINFO data structure (C/C++ API)**

Field	Description
SerialNumber	Serial number of the Saturn SX1 FPGA module

ProductNumber	Product number of the Saturn SX1 FPGA module
FirmwareVersionMaj	Major revision number of the Saturn SX1 FPGA module's FX2 firmware
FirmwareVersionMin	Minor revision number of the Saturn SX1 FPGA module's FX2 firmware
PicVersionMaj	Major revision number of the Saturn SX1 FPGA module's PIC firmware
PicVersionMin	Minor revision number of the Saturn SX1 FPGA module's PIC firmware
FPGAType	Type of the Saturn SX1 FPGA module's FPGA
DDR2Size	Size of the Saturn SX1 FPGA module's DDR2 SDRAM in 32 MB steps
SSRAMSize	Size of the Saturn SX1 FPGA module's SSRAM in 128 KB steps
FlashSize	Size of the Saturn SX1 FPGA module's SPI FLASH in 1 MB steps

**Table 15: ENCFX2\_CARDINFO data structure (C/C++ API)**

### 3.1.4.2 ENCFX2\_STATS

This data structure is used to hold some data transfer statistics.

```
typedef struct {
    unsigned long long BytesSent;
    unsigned long long BytesReceived;
} ENCFX2_STATS;
```

**Code Snippet 7: ENCFX2\_STATS data structure (C/C++ API)**

Field	Description
BytesSent	Bytes sent to the Saturn SX1 FPGA module since the module is open
BytesReceived	Bytes received from the Saturn SX1 FPGA module since the module is open

**Table 16: ENCFX2\_STATS data structure (C/C++ API)**

## 3.1.5 Functions

### 3.1.5.1 EncFX2ListCards()

This function is used to discover all Saturn SX1 FPGA modules connected to a host PC and gather their general module configuration information.

```

ENCXF2_STATUS ENC_EXPORT EncFX2ListCards( unsigned int*   Num,
                                           ENCFX2_CARDINFO* CardInfos,
                                           unsigned int   MaxInfos );

```

**Code Snippet 8: EncFX2ListCards() function prototype (C/C++ API)**

Parameter	Direction	Description
Num	Out	The total number of Saturn SX1 FPGA modules available ( $\leq$ MaxInfos)
CardInfos	Out	Pointer to an array of ENCFX2_CARDINFO structures. The array contains the general module configuration information for each available module.
MaxInfos	In	0: All available Saturn SX1 FPGA modules are reported >0: Only the first <i>MaxInfos</i> Saturn SX1 FPAG modules are reported

**Table 17: EncFX2ListCards() function parameters (C/C++ API)**

### 3.1.5.2 EncFX2OpenCard()

This function is used to open a Saturn SX1 FPGA module specified by product and serial number. Product and serial numbers of the Saturn SX1 FPGA modules connected to the host PC may be retrieved by calling the EncFx2ListCards() function.

```

ENCXF2_STATUS ENC_EXPORT EncFX2OpenCard( unsigned int   ProductNumber,
                                           unsigned int   SerialNumber,
                                           ENCFX2_HANDLE* Handle );

```

**Code Snippet 9: EncFX2OpenCard() function prototype (C/C++ API)**

Parameter	Direction	Description
ProductNumber	In	Product number of the Saturn SX1 FPGA module to be opened
SerialNumber	In	Serial number of the Saturn SX1 FPGA module to be opened
Handle	Out	Handle to the opened Saturn SX1 FPGA module

**Table 18: EncFX2OpenCard () function parameters (C/C++ API)**

### 3.1.5.3 EncFX2CloseCard()

This function is used to close a Saturn SX1 FPGA module previously opened by a call to the EncFx2OpenCard() function.

```

ENCXF2_STATUS ENC_EXPORT EncFX2CloseCard( ENCFX2_HANDLE Handle );

```

**Code Snippet 10: EncFX2CloseCard() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to be closed

**Table 19: EncFX2CloseCard() function parameters (C/C++ API)**

### 3.1.5.4 EncFX2GetCardInfo()

This function is used to gather the general module configuration information of an already opened Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2GetCardInfo( ENCFX2_HANDLE Handle,
                                             ENCFX2_CARDINFO* Info );
```

**Code Snippet 11: EncFX2GetCardInfo() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to get the information from
Info	Out	Pointer to an ENCFX2_CARDINFO structure. After the function returned the structure contains the general module configuration information for the requested module.

**Table 20: EncFX2GetCardInfo() function parameters (C/C++ API)**

### 3.1.5.5 EncFX2GetCardstateFlag()

This function is used to get the flag value of a specific monitoring parameter of a Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2GetCardstateFlag( ENCFX2_HANDLE Handle,
                                                  ENCFX2_CARDSTATE State,
                                                  ENCFX2_CARDSTATE_FLAG* Flag );
```

**Code Snippet 12: EncFX2GetCardstateFlag() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to get the status flags from
State	In	An ENCFX2_CARDSTATE value defining which parameter flag should be retrieved
Flag	Out	Pointer to an ENCFX2_CARDSTATE_FLAG variable. After the function returned, the variable holds the retrieved parameter flag value.

**Table 21: EncFX2GetCardstateFlag() function parameters (C/C++ API)**

### 3.1.5.6 EncFX2GetCardstateValue()

This function is used to get the value of a specific monitoring parameter of a Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2GetCardstateValue( ENCFX2_HANDLE Handle,
                                                  ENCFX2_CARDSTATE State,
                                                  int* Value );
```

**Code Snippet 13: EncFX2GetCardstateValue() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to get the status value from
State	In	An ENCFX2_CARDSTATE value defining which parameter value should be retrieved

Value	Out	Pointer to an integer variable. After the function returned, the variable holds the retrieved parameter value. Temperatures are represented in °C while voltages are represented in mV.
-------	-----	---

**Table 22: EncFX2GetCardstateValue() function parameters (C/C++ API)**

### 3.1.5.7 EncFX2GetErrorMessage()

This function is used to get the error message string corresponding to an ENCFX2\_STATUS value.

```
ENCFX2_STATUS ENC_EXPORT EncFX2GetErrorMessage( ENCFX2_STATUS Status,
char** Buffer );
```

**Code Snippet 14: EncFX2GetErrorMessage() function prototype (C/C++ API)**

Parameter	Direction	Description
Status	In	The status code for which the message should be retrieved
Buffer	Out	Pointer to a char array. After the function returned, the char array holds the error message string.

**Table 23: EncFX2GetErrorMessage() function parameters (C/C++ API)**

### 3.1.5.8 EncFX2Configure()

This function is used to configure the FPGA of a Saturn SX1 FPGA module with a bitstream which is available in memory.

```
ENCFX2_STATUS ENC_EXPORT EncFX2Configure( ENCFX2_HANDLE Handle,
const unsigned char* Image,
const unsigned int Length );
```

**Code Snippet 15: EncFX2Configure() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to be configured
Image	In	Byte array containing the FPGA bitstream
Length	In	Length of the FPGA bitstream in bytes

**Table 24: EncFX2Configure() function parameters (C/C++ API)**

### 3.1.5.9 EncFX2ConfigureFromFile()

This function is used to configure the FPGA of a Saturn SX1 FPGA module with a bitstream which is available as a .bit file.

```
ENCFX2_STATUS ENC_EXPORT EncFX2ConfigureFromFile( ENCFX2_HANDLE Handle,
char* Filename );
```

**Code Snippet 16: EncFX2ConfigureFromFile() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to be configured

Filename	In	Char array holding the filename of the FPGA bitstream (including ".bit")
----------	----	--

**Table 25: EncFX2ConfigureFromFile() function parameters (C/C++ API)**

### 3.1.5.10 EncFX2GetStats()

This function is used to get data transfer statistics for a Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2GetStats( ENCFX2_HANDLE Handle,
                                          ENCFX2_STATS* Stats );
```

**Code Snippet 17: EncFX2GetStats() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to get the statistics from
Stats	Out	Pointer to a ENCFX2_STATS data structure. After the function returned, the structure holds the statistics values.

**Table 26: EncFX2GetStats() function parameters (C/C++ API)**

### 3.1.5.11 EncFX2SetTimeout()

This function is used to set the read and write operation timeouts. EncFX2Poll() has its own timeout given as a parameter.

```
ENCFX2_STATUS ENC_EXPORT EncFX2SetTimeout( ENCFX2_HANDLE Handle,
                                           unsigned int ReadMilliseconds,
                                           unsigned int WriteMilliseconds );
```

**Code Snippet 18: EncFX2SetTimeout() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to get the statistics from
ReadMilliseconds	In	The timeout for read operations using EncFX2Read() to be set in milliseconds. A timeout of 0 waits forever.
WriteMilliseconds	In	The timeout for write operations using EncFX2Write() to be set in milliseconds. A timeout of 0 waits forever.

**Table 27: EncFX2SetTimeout () function parameters (C/C++ API)**

### 3.1.5.12 EncFX2Write()

This function is used to write data to a Saturn SX1 FPGA module. It calls EncFX2QueueWriteBegin(), EncFX2QueueWriteData() and EncFX2FlushQueue().

```
ENCFX2_STATUS ENC_EXPORT EncFX2Write( ENCFX2_HANDLE Handle,
                                       unsigned int Address,
                                       unsigned int Length,
                                       unsigned char* Data );
```

**Code Snippet 19: EncFX2Write() function prototype (C/C++ API)**

Parameter	Direction	Description
-----------	-----------	-------------

Handle	In	Handle to the Saturn SX1 FPGA module to write data to
Address	In	Byte address to write the data to (offset into the 4 GB Saturn SX1 FPGA module's ClustraBus address space)
Length	In	Number of bytes to write to the Saturn SX1 FPGA module. Length = 0 is interpreted as Length = 4 GB.
Data	In	Byte array holding the data to be written to the Saturn SX1 FPGA module.

**Table 28: EncFX2Write() function parameters (C/C++ API)**

### 3.1.5.13 EncFX2QueueWriteBegin()

This function is used to initiate a write transfer to a Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2QueueWriteBegin( ENCFX2_HANDLE Handle,
                                                unsigned int Address,
                                                unsigned int Length );
```

**Code Snippet 20: EncFX2QueueWriteBegin() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to write data to
Address	In	Byte address to write the data to (offset into the 4 GB Saturn SX1 FPGA module's ClustraBus address space)
Length	In	Number of bytes to write to the Saturn SX1 FPGA module. Length = 0 is interpreted as Length = 4 GB.

**Table 29: EncFX2QueueWriteBegin() function parameters (C/C++ API)**

### 3.1.5.14 EncFX2QueueWriteData()

This function is used to add write data to an already initiated write transfer to a Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2QueueWriteData( ENCFX2_HANDLE Handle,
                                                unsigned char* Data,
                                                unsigned int Length );
```

**Code Snippet 21: EncFX2QueueWriteData() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to write data to
Data	In	Byte array holding the data to be added to the write transfer
Length	In	Number of bytes contained in the Data byte array

**Table 30: EncFX2QueueWriteData() function parameters (C/C++ API)**

### 3.1.5.15 EncFX2Read()

This function is used to read data from a Saturn SX1 FPGA module. It calls EncFX2QueueRequestData(), EncFX2FlushQueue() and EncFX2Poll().

```
ENCFX2_STATUS ENC_EXPORT EncFX2Read( ENCFX2_HANDLE Handle,
                                     unsigned int  Address,
                                     unsigned int  Length,
                                     unsigned char* Data,
                                     unsigned int  DataLength );
```

**Code Snippet 22: EncFX2Read() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to read data from
Address	In	Byte address to read the data from (offset into the 4 GB Saturn SX1 FPGA module's ClustraBus address space)
Length	In	Number of bytes to read from the Saturn SX1 FPGA module. Length = 0 is interpreted as Length = 4 GB.
Data	Out	Byte array for receiving the data to be read from the Saturn SX1 FPGA module.
DataLength	In	Length of the Data byte array in bytes

**Table 31: EncFX2Read() function parameters (C/C++ API)**

### 3.1.5.16 EncFX2QueueRequestData()

This function is used to request data from a Saturn SX1 FPGA module.

```
ENCFX2_STATUS ENC_EXPORT EncFX2QueueRequestData( ENCFX2_HANDLE Handle,
                                                  unsigned int  Address,
                                                  unsigned int  Length );
```

**Code Snippet 23: EncFX2QueueRequestData() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to read data from
Address	In	Byte address to read the data from (offset into the 4 GB Saturn SX1 FPGA module's ClustraBus address space)
Length	In	Number of bytes to read from the Saturn SX1 FPGA module. Length = 0 is interpreted as Length = 4 GB.

**Table 32: EncFX2QueueRequestData() function parameters (C/C++ API)**

### 3.1.5.17 EncFX2FlushQueue()

This function is used to commit all read and write transfers which have already been added to a Saturn SX1 FPGA module's queue.

```
ENCFX2_STATUS ENC_EXPORT EncFX2FlushQueue( ENCFX2_HANDLE Handle );
```

**Code Snippet 24: EncFX2FlushQueue() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to flush the queue

**Table 33: EncFX2FlushQueue() function parameters (C/C++ API)**

### 3.1.5.18 EncFX2Poll()

This function is used to poll for messages received from a Saturn SX1 FPGA module. It is used to accomplish regular reads initiated by EncFX2QueueRequestData() as well as to retrieve data received by DMA transfers initiated by a Saturn SX1 FPGA module.

```

ENCFX2_STATUS ENC_EXPORT EncFX2Poll( ENCFX2_HANDLE Handle,
                                     unsigned int* Address,
                                     unsigned int* Length,
                                     unsigned char* Buffer,
                                     unsigned int BufferLength,
                                     unsigned int Timeout );

```

**Code Snippet 25: EncFX2Poll() function prototype (C/C++ API)**

Parameter	Direction	Description
Handle	In	Handle to the Saturn SX1 FPGA module to read data from
Address	Out	Byte address corresponding to the first byte returned in the Buffer char array (offset into the 4 GB Saturn SX1 FPGA module's ClustraBus address space, is returned as zero in the event of a poll timeout)
Length	Out	Number of bytes written to the Buffer char array (is returned as zero in the event of a poll timeout)
Buffer	Out	Char array for receiving the polled data.
BufferLength	In	Size of the Buffer char array in bytes
Timeout	In	Timeout of the operation in ms

**Table 34: EncFX2Poll() function parameters (C/C++ API)**

## 3.2 Java

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## 3.3 .NET

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## **3.4 MATLAB**

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## **3.5 LabView**

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

# 4 Reference Design

## 4.1 FPGA Firmware

### 4.1.1 Functional Description

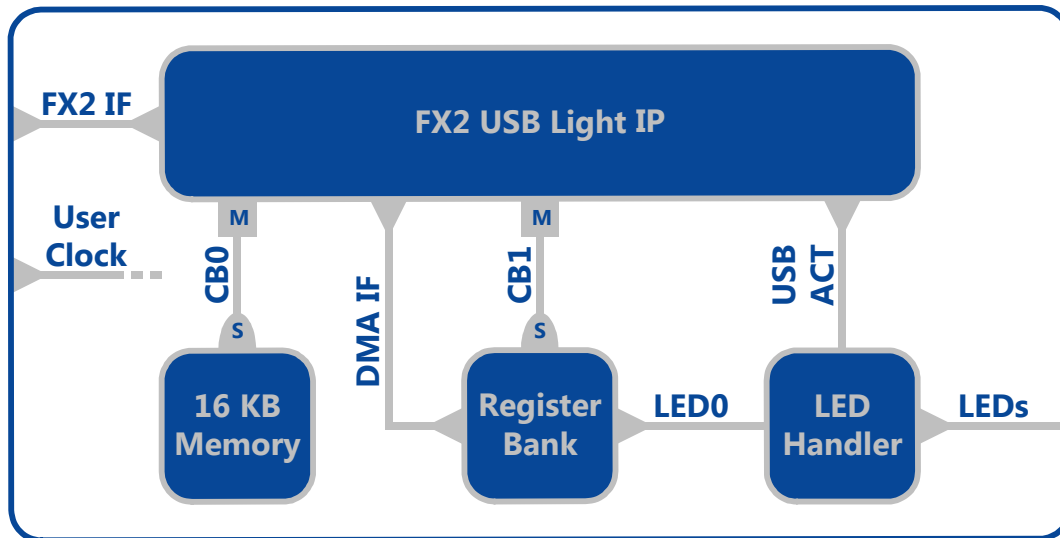


Figure 9: Reference design block diagram

The FX2 USB Light IP reference design consists of the FX2 USB Light IP core, 16 KB of FPGA-internal memory on ClustraBus port 0, a register bank for DMA and LED control on ClustraBus port 1, as well as some blinking LED logic.

The reference design FPGA firmware allows reading/writing from/to the 16 KB memory block, initiating DMA transfers via the DMA request FIFO within the register bank and controlling a LED via the register bank.

### 4.1.2 Design Hierarchy

Entity Name	Description
fx2_usb_light_rd	Reference design top level
fx2_usb_light	FX2 USB Light IP netlist wrapper
fx2_usb_light_nl	FX2 USB Light IP netlist
fx2_usb_light_rd_rb	Reference design register bank
cb_register_bank	ClustraBus register bank core <sup>3</sup>
cgn_fifo_64_30	Xilinx Core Generator FIFO (64 x 30)
fx2_usb_light_rd_mem	Reference design BRAM memory with ClustraBus interface

**Table 35: Reference design hierarchy**

## 4.1.3 Interfaces

### 4.1.3.1 User Clock Interface

Signal Name	Width	Direction	Description
ClkUser0	1	In	User clock input (24 MHz)

**Table 36: Reference design user clock interface description**

**Attention:** The reference design is configured to work with the Saturn SX1 factory PLL settings (FX2 clock frequency = 48 MHz, user clock frequency = 24 MHz).

### 4.1.3.2 FX2 Interface

Please refer to section 2.2.3 on page 11 for a description of the FX2 interface.

### 4.1.3.3 LED Interface

Signal Name	Width	Direction	Description
Led_N	4	Out	Saturn SX1 LED outputs (active low)

**Table 37: Reference design LED interface description**

## 4.1.4 Memory Map

Table 38 shows the memory map of the reference design. Please refer to section 6 for conventions for register descriptions.

Byte Address	Dword Address	Size	CB Slave	Region Name	Type	Description
0x00000000	0x00000000	16 KB	0	MEMORY	RW	BRAM memory
0x00004000	0x00001000	2 GB – 16 KB	0	-	R0	Reserved
0x80000000	0x20000000	4 B	1	REG_IP_REV	RW	IP revision register
0x80000004	0x20000001	4 B	1	REG_RD_REV	RW	Reference design revision register
0x80000008	0x20000002	4 B	1	REG_LED_CTL	RW	LED control register
0x8000000C	0x20000003	4 B	1	REG_DMA_FIFO	W	DMA FIFO register
0x80000010	0x20000004	2 GB – 16 B	1	-	R0	Reserved

**Table 38: Reference design memory map**

## 4.1.5 Register description

### 4.1.5.1 IP Revision Register (REG\_IP\_REV)

Bit	Size	Field Name	Type	Reset	Description
31:16	16	-	R0	0	Reserved
15:8	8	MAJ	R	0	Major revision number (0..255)
7:0	8	MIN	R	0	Minor revision number (00..99)

Table 39: Reference design IP revision register (REG\_IP\_REV)

### 4.1.5.2 Reference Design Revision Register (REG\_RD\_REV)

Bit	Size	Field Name	Type	Reset	Description
31:16	16	-	R0	0	Reserved
15:8	8	MAJ	R	0	Major revision number (0..255)
7:0	8	MIN	R	0	Minor revision number (00..99)

Table 40: Reference design revision register (REG\_RD\_REV)

### 4.1.5.3 LED Control Register (REG\_LED\_CTL)

Bit	Size	Field Name	Type	Reset	Description
31:1	31	-	R0	0	Reserved
0	1	LED0	R	0	0: Saturn SX1 LED 0 off 1: Saturn SX1 LED 0 on

Table 41: Reference design LED control register (REG\_LED\_CTRL)

### 4.1.5.4 DMA FIFO Register (REG\_DMA\_FIFO)

The DMA FIFO register address provides access to the write port of the DMA FIFO located in the user design register bank – it is therefore a read-only address.

In order to issue a DMA transfer, a DMA start address and a DMA length have to be written to the DMA FIFO. The DMA start address has to be written first.

The DMA controller FSM located in the reference design register bank then reads the DMA parameters from the DMA FIFO and requests a DMA transfer via the DMA interface of the FX2 USB light IP core.

The DMA FIFO is able to hold a maximum of 32 DMA transfers, meaning 32 address/length pairs.

**Attention:** There is no DMA FIFO overflow detection in the reference design FPGA firmware. The host software is thus in charge of preventing a DMA FIFO overflow.

Table 42 shows the DMA FIFO address format while Table 43 shows the DMA FIFO length format.

**Attention:** Please note that the DMA address and length values must be dword-aligned, meaning they must be multiples of 4 bytes.

Bit	Size	Field Name	Type	Reset	Description
31:0	32	ADDR	W	-	DMA start byte address, dword-aligned (bits 1:0 are ignored)

**Table 42: DMA FIFO address format (REG\_DMA\_FIFO\_ADDR)**

Bit	Size	Field Name	Type	Reset	Description
31:0	32	LENGTH	W	-	DMA length in bytes, dword-aligned (bits 1:0 are ignored)

**Table 43: DMA FIFO length format (REG\_DMA\_FIFO\_LENGTH)**

## 4.1.6 LED Activity Interpretation

The four LEDs of the Saturn SX1 FPGA module are used to indicate general status and access information. Table 44 shows how to interpret the LED activity.

LED Number	Description
0	Controlled by the LED control register (REG_LED_CTL) bit 0.
1	Shows a 0.125 s flash each time there is an internal ClustraBus access.
2	Blinks with 0.125 s period if the FX2 interface clock (Fx2_IfClk, 48 MHz) is present.
3	Blinks with 0.25 s period if the user clock (ClkUser0, 24 MHz) is present.

**Table 44: Reference design LED activity interpretation**

## 4.1.7 Files

File Name	Directory	Description
fx2_usb_light.vhd	.\ip	Please see Table 8 on page 14 for a description
fx2_usb_light_nl.ngc	.\ip	Please see Table 8 on page 14 for a description
fx2_usb_light.ucf	.\ip	Please see Table 8 on page 14 for a description
fx2_usb_light_rd.ise	.\ise	Xilinx ISE project file
xc3sd3400a-5csg484c.cgp	.\cgn	Xilinx Core Generator project file
cgn_bram_4k_32.ngc	.\cgn	Core Generator BRAM (4k x 32) netlist
cgn_bram_4k_32.vhd	.\cgn	Core Generator BRAM (4k x 32) simulation model
cgn_bram_4k_32.xco	.\cgn	Core Generator BRAM (4k x 32) core description
cgn_fifo_64_30.ngc	.\cgn	Core Generator FIFO (64 x 30) netlist

cgn_fifo_64_30.vhd	.\cgn	Core Generator FIFO (64 x 30) simulation model
cgn_fifo_64_30.xco	.\cgn	Core Generator FIFO (64 x 30) core description
cb_register_bank.vhd	.\src	ClustraBus Register Bank
cb_register_bank_pkg.vhd	.\src	ClustraBus Register Bank Package
fx2_usb_light_rd_rb.vhd	.\src	Reference design register bank
fx2_usb_light_rd_mem.vhd	.\src	Reference design BRAM memory file
fx2_usb_light_rd.vhd	.\src	Reference design top-level file
fx2_usb_light_rd.ucf	.\ucf	Reference design user constraints file

**Table 45: Reference design FPGA firmware files**

## 4.1.8 Development Tools

The FX2 USB IP Light IP reference design has been created with Xilinx ISE 11.3.

**Attention:** The “Add I/O Buffers” option of the Xilinx XST synthesis tool must be checked (meaning turned on) for the reference design to compile properly.

## 4.2 Host PC Software

### 4.2.1 C / C++

#### 4.2.1.1 Module Status and LED Demo (status\_led)

This demo shows how to list and open a Saturn SX1 FPGA module connected to the host PC. It also demonstrates how to retrieve the module’s temperature and voltage monitoring status. In addition, allows controlling a LED by writing to a register located within the FPGA’s reference design firmware.

#### 4.2.1.2 Random Read/Write Demo (read\_demo)

This demo shows how a host PC transfers data from and to a Saturn SX1 FPGA module. It first initializes the memory located in the FPGA’s reference design firmware with random values. Afterwards, a read/write test with random transfer type (r/w), random transfer length and random write data is performed.

#### 4.2.1.3 Random DMA Demo (dma\_demo)

This demo shows how a Saturn SX1 FPGA module sends data to a host PC via DMA. It is basically the same like the Random Read/Write Demo except that the read transfers are replaced by DMA transfers initiated via the FPGA’s reference design register bank. In a real-world design, the FPGA would of course initiate the DMA transfers itself, driven by e.g. the reception of streaming data.

#### 4.2.1.4 Development Tools

The MinGW 5.1.4 C compiler<sup>4</sup> is used to build the MinGW host PC software, while Microsoft Visual Studio 2008 Express has been used to compile the VisualC++ host PC Software.

## **4.2.2 Java**

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## **4.2.3 .NET**

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## **4.2.4 MATLAB**

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

## **4.2.5 LabView**

Please contact Enclustra for further information. Contact information is provided on [www.enclustra.com](http://www.enclustra.com).

# 5 Use Cases

## 5.1 Acquiring and Processing Data from a Streaming Data Source

### 5.1.1 Overview

This use case covers systems designs where a data is acquired from a streaming data source, for example an ADC running at a constant sample rate.

Table 46 lists the basic design parameters used throughout the use case.

Parameter	Description	Example Value
DATA_RATE	Data rate of the streaming data source	2.5 MB/s
POLL_PERIOD	Time between two subsequent calls to EncFX2Poll() in the Host PC software	100 ms
DMA_LENGTH	Length of one DMA transfer from the FPGA to the host PC	250 KB
DMA_BASE_ADDR	Start address for DMA transfers	0x00000000
FIFO_SIZE	Size of the data acquisition FIFO	500 KB

**Table 46: Acquiring and Processing Data from a Streaming Data Source: Design Parameters**

The following fundamental relations between data rate, poll period, DMA length and FIFO size do apply:

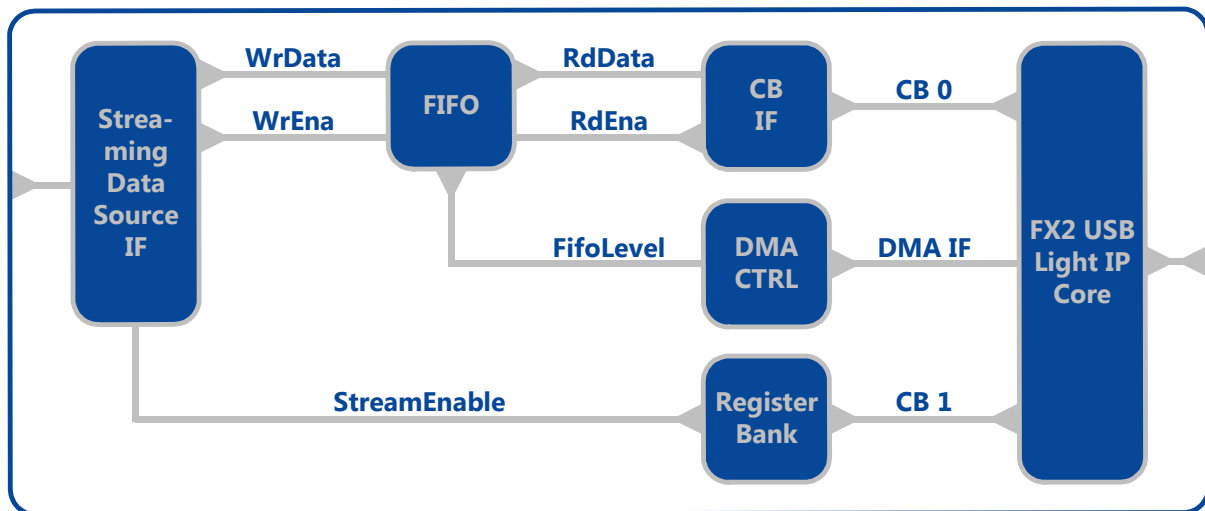
$$(1) \text{POLL\_PERIOD} \leq \frac{\text{DMA\_LENGTH}}{\text{DATA\_RATE}}$$

$$(2) \text{FIFO\_SIZE} \geq \text{DMA\_LENGTH}$$

These equations assume a constant data rate for the streaming data source. If this is not the case, the FIFO size must be increased in order to compensate for temporary peaks in the streaming data rate.

It is obvious that for the chosen example values (see Table 46) the FIFO size is too big to be implemented with FPGA-internal memory. This problem can either be handled by decreasing the poll period or by using the FPGA-external SRAM available on the Saturn SX1 FPGA module for the FIFO memory.

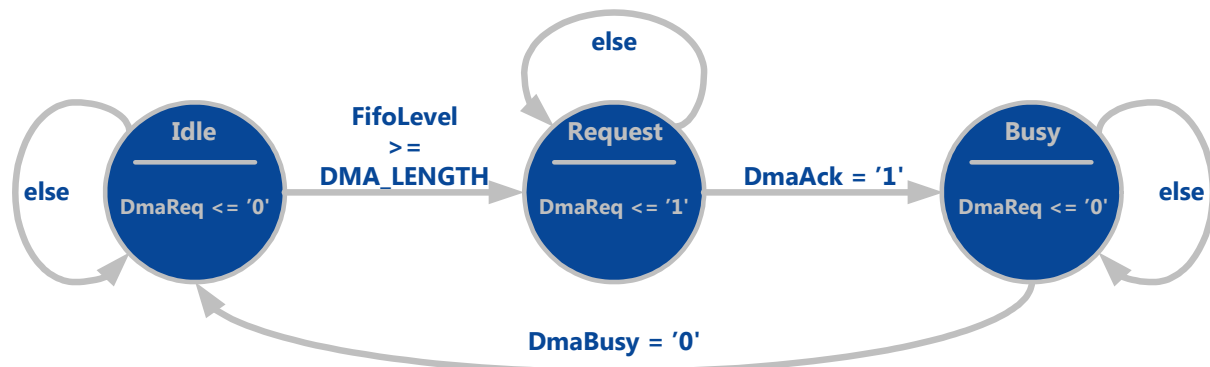
## 5.1.2 FPGA Firmware



**Figure 10: Acquiring and Processing Data from a Streaming Data Source: FPGA Firmware**

Figure 10 shows the proposed FPGA firmware architecture. It consists of an interface block to the streaming data source, the data acquisition FIFO (implemented with either internal or external memory) with a ClustraBus interface, the DMA controller and an optional ClustraBus register bank.

Data received from the streaming data source is written to the data acquisition FIFO. As soon as there is enough data available inside the FIFO ( $\text{FifoLevel} \geq \text{DMA\_LENGTH}$ ), the DMA controller requests a DMA transfer via the FX2 USB Light IP core's DMA interface (see section 2.2.5 on page 13 for details). The DMA base address and DMA length are constant in this use case ( $\text{DmaAddr} = \text{DMA\_BASE\_ADDR}$ ,  $\text{DmaLen} = \text{DMA\_LENGTH}$ ).<sup>5</sup> After the DMA request has been acknowledged, the DMA controller waits for completion of the DMA transfer and goes back to idle afterwards. Figure 11 shows the DMA controller FSM state diagram.



**Figure 11: Acquiring and Processing Data from a Streaming Data Source: DMA Controller FSM State Diagram**

A DMA request from the DMA controller is handled by the DMA engine located within the FX2 USB Light IP core. Upon acknowledging a DMA request, the DMA engine starts to read the data via ClustraBus, starting at the DMA base address. The data read via ClustraBus is then transferred to the FX2 device's transmit FIFOs, from where it can be read by the host PC software.

<sup>5</sup> The DMA base address is equivalent to the data acquisition FIFO base address within the ClustraBus address space (0x00000000 in this example).

### 5.1.3 Host PC Software

Code Snippet 26 shows a pseudo-code example of a basic data acquisition and processing loop. The host PC software polls for data until there is a message available. It then checks the message for correct base address and length values.<sup>6</sup> It then processes the received data and goes back to polling afterwards.

```
while (Running)
{
    // wait for DMA message
    while (Address == 0 && Length == 0)
    {
        EncFX2Poll(Handle, &Address, &Length, &Buffer, POLL_TIMEOUT);
    }

    // check address and length to identify a DMA message
    if (Address == DMA_BASE_ADDR && Length = DMA_LENGTH)
    {
        // process received data
        process_data(Buffer);
    }
    else
    {
        // handle other messages
        handle_other_messages();
    }
}
```

#### Code Snippet 26: Acquiring and Processing Data from a Streaming Data Source: Host PC Software

This approach of acquiring data from a streaming data source works without any additional synchronization overhead. However, it requires that the time it takes for transferring and processing the data is smaller than the maximum allowed poll period.

---

<sup>6</sup> This is a way to detect whether the received message is the wanted DMA message or some other message which could have been requested by other parts of the host PC software.

# 6 Conventions for Register Descriptions

## 6.1 Register Field Types

Field Type	Description
R	Read only. This bit can only be modified by hardware, writing to this bit has no effect.
R0	Read only (always reads as zero). Writing to this bit has no effect
R1	Read only (always reads as one). Writing to this bit has no effect.
RCW1	Read only, cleared by write one. A readable status bit that can be cleared by writing a one to it.
RE	Read only with side effect. Reading from this bit has an effect on the hardware (e.g. reading from a FIFO)
RW	Standard read/write bit. Only software or a hardware reset can change the bit's value.
RWV	Volatile read/write bit. This value of this bit may be changed by software and also generally by hardware, not only through a hardware reset.
W	Write only.
WE	Write only with side effect. Writing to this bit has an additional effect on the hardware (e.g. writing to a FIFO)
W1R0	Write one read zero. Writing a one to this bit has an effect on the hardware, but the bit always reads as zero ('self-clearing bit').

Table 47: Register Field Types

## 6.2 Reset Values

Reset Value	Description
n	Resets to n ( $n = 0..2^{\text{FIELD\_SIZE}} - 1$ ). Prefixes define the number format (reset value zero and reset values for single bit fields do not need to be prefixed):  d: Decimal

	b: Binary 0x: Hexadecimal
-	Undefined at reset.
U	Unaffected by reset.
[Identifier]	Reset value is determined by the value of the denoted identifier, which may be a signal, a generic or a constant.

**Table 48: Reset Values**

## 6.3 Identifiers

A unique identifier for a register field (e.g. for the use as constants in SW code) is thought to be composed of region, register and field names, separated with underscores. For the FPGA I major firmware revision register field for example this would result in REG\_FWREVI\_MAJ.

## 6.4 Size and Address Information

Size and address information in address maps (e.g. base addresses or address offsets) is generally declared in units of bytes.

Size information in register or data format descriptions is generally declared in units of bits.

## 6.5 Number Formats

### 6.5.1 Fixed Point

Fixed-Point number formats are specified in the form **[s,i,f]** with the following meanings for s, i and f:

- **s**: Sign bit (0: unsigned, 1: signed)
- **i**: Integer bit count
- **f**: Fractional bit count

The total width of a fixed point number is thus given by **w=s+i+f**. Unless otherwise noted, signed numbers are represented in 2's complement.

### 6.5.2 Floating Point

Floating Point number formats are specified in the form **[e,s]** with the following meanings for e and s:

- **e**: Exponent bit count
- **s**: Significand bit count

The total width of a floating point number is thus given by **w=e+s+1** due to the fact that floating point numbers always include a sign bit.

The commonly used IEEE standard floating point formats may also be specified by the following notion:

- **FPS:** Single precision floating point [8,23]
- **FPD:** Double precision floating point [11,52]

## Figures

Figure 1: Simplified system layering.....	5
Figure 2: Installation directory structure.....	6
Figure 3: Windwos driver installation screenshot 1.....	7
Figure 4: Windwos driver installation screenshot 2.....	8
Figure 5: Windwos driver installation screenshot 3.....	8
Figure 6: Windwos driver installation screenshot 4.....	9
Figure 7: IP core block diagram.....	10
Figure 8: IP core DMA interface waveforms.....	13
Figure 9: Reference design block diagram .....	28
Figure 10: Acquiring and Processing Data from a Streaming Data Source: FPGA Firmware.....	35
Figure 11: Acquiring and Processing Data from a Streaming Data Source: DMA Controller FSM State Diagram.....	35

## Tables

Table 1: IP core user clock and reset interface description.....	10
Table 2: IP core FX2 clock and reset interface description.....	11
Table 3: IP core external FX2 interface description .....	11
Table 4: IP core ClustraBus interface description .....	12
Table 5: IP core ClustraBus port address ranges.....	13
Table 6: IP core DMA interface description.....	13
Table 7: IP core IP revision interface description.....	14
Table 8: IP core files .....	14
Table 9: IP core resource usage.....	14
Table 10: C/C++ API files.....	15
Table 11: ENCFX2_STATUS enumerated type (C/C++ API).....	17
Table 12: ENCFX2_FPGATYPE enumerated type (C/C++ API).....	17
Table 13: ENCFX2_CARDSTATE enumerated type (C/C++ API) .....	18
Table 14: ENCFX2_CARDSTATE_FLAG enumerated type (C/C++ API).....	18
Table 15: ENCFX2_CARDINFO data structure (C/C++ API).....	19
Table 16: ENCFX2_STATS data structure (C/C++ API).....	19
Table 17: EncFX2ListCards() function parameters (C/C++ API).....	20

Table 18: EncFX2OpenCard () function parameters (C/C++ API) .....	20
Table 19: EncFX2CloseCard() function parameters (C/C++ API).....	20
Table 20: EncFX2GetCardInfo() function parameters (C/C++ API).....	21
Table 21: EncFX2GetCardstateFlag() function parameters (C/C++ API) .....	21
Table 22: EncFX2GetCardstateValue() function parameters (C/C++ API) .....	22
Table 23: EncFX2GetErrorMessage() function parameters (C/C++ API) .....	22
Table 24: EncFX2Configure() function parameters (C/C++ API) .....	22
Table 25: EncFX2ConfigureFromFile() function parameters (C/C++ API).....	23
Table 26: EncFX2GetStats() function parameters (C/C++ API) .....	23
Table 27: EncFX2SetTimeout () function parameters (C/C++ API).....	23
Table 28: EncFX2Write() function parameters (C/C++ API) .....	24
Table 29: EncFX2QueueWriteBegin() function parameters (C/C++ API) .....	24
Table 30: EncFX2QueueWriteData() function parameters (C/C++ API) .....	24
Table 31: EncFX2Read() function parameters (C/C++ API) .....	25
Table 32: EncFX2QueueRequestData() function parameters (C/C++ API).....	25
Table 33: EncFX2FlushQueue() function parameters (C/C++ API) .....	26
Table 34: EncFX2Poll() function parameters (C/C++ API).....	26
Table 35: Reference design hierarchy .....	29
Table 36: Reference design user clock interface description.....	29
Table 37: Reference design LED interface description .....	29
Table 38: Reference design memory map.....	29
Table 39: Reference design IP revision register (REG_IP_REV) .....	30
Table 40: Reference design revision register (REG_RD_REV) .....	30
Table 41: Reference design LED control register (REG_LED_CTRL) .....	30
Table 42: DMA FIFO address format (REG_DMA_FIFO_ADDR) .....	31
Table 43: DMA FIFO length format (REG_DMA_FIFO_LENGTH) .....	31
Table 44: Reference design LED activity interpretation .....	31
Table 45: Reference design FPGA firmware files .....	32
Table 46: Acquiring and Processing Data from a Streaming Data Source: Design Parameters .....	34
Table 47: Register Field Types.....	37
Table 48: Reset Values .....	38

## Code Snippets

Code Snippet 1: ENCFX2_HANDLE scalar type (C/C++ API) .....	15
Code Snippet 2: ENCFX2_STATUS enumerated type (C/C++ API) .....	16
Code Snippet 3: ENCFX2_FPGATYPE enumerated type (C/C++ API) .....	17
Code Snippet 4: ENCFX2_CARDSTATE enumerated type (C/C++ API) .....	17
Code Snippet 5: ENCFX2_CARDSTATE_FLAG enumerated type (C/C++ API) .....	18
Code Snippet 6: ENCFX2_CARDINFO data structure (C/C++ API) .....	18
Code Snippet 7: ENCFX2_STATS data structure (C/C++ API) .....	19
Code Snippet 8: EncFX2ListCards() function prototype (C/C++ API) .....	20
Code Snippet 9: EncFX2OpenCard() function prototype (C/C++ API) .....	20
Code Snippet 10: EncFX2CloseCard() function prototype (C/C++ API) .....	20
Code Snippet 11: EncFX2GetCardInfo() function prototype (C/C++ API) .....	21
Code Snippet 12: EncFX2GetCardstateFlag() function prototype (C/C++ API) .....	21
Code Snippet 13: EncFX2GetCardstateValue() function prototype (C/C++ API) .....	21
Code Snippet 14: EncFX2GetErrorMessage() function prototype (C/C++ API) .....	22
Code Snippet 15: EncFX2Configure() function prototype (C/C++ API) .....	22
Code Snippet 16: EncFX2ConfigureFromFile() function prototype (C/C++ API) .....	22
Code Snippet 17: EncFX2GetStats() function prototype (C/C++ API) .....	23
Code Snippet 18: EncFX2SetTimeout() function prototype (C/C++ API) .....	23
Code Snippet 19: EncFX2Write() function prototype (C/C++ API) .....	23
Code Snippet 20: EncFX2QueueWriteBegin() function prototype (C/C++ API) .....	24
Code Snippet 21: EncFX2QueueWriteData() function prototype (C/C++ API) .....	24
Code Snippet 22: EncFX2Read() function prototype (C/C++ API) .....	25
Code Snippet 23: EncFX2QueueRequestData() function prototype (C/C++ API) .....	25
Code Snippet 24: EncFX2FlushQueue() function prototype (C/C++ API) .....	25
Code Snippet 25: EncFX2Poll() function prototype (C/C++ API) .....	26
Code Snippet 26: Acquiring and Processing Data from a Streaming Data Source: Host PC Software ....	36

## References

---

- <sup>1</sup> *ClustraBus Interconnects*, Enclustra GmbH, 2009, <http://www.enclustra.com/CbIntercon>
- <sup>2</sup> *ClustraBus Protocol Specification*, Enclustra GmbH, 2009, <http://www.enclustra.com/CbSpec>
- <sup>3</sup> *ClustraBus Register Bank*, Enclustra GmbH, 2009, <http://www.enclustra.com/CbMisc>
- <sup>4</sup> *Minimalist GNU for Windows*, <http://www.mingw.org/>