

SMIMS Engine Software Development Kits

User Guide

VLFD- USB - FDP3P7

Version 1.1.0

SMIMS Rapid FPGA Platform
VLFD-USB-FDP3P7

Target FPGA : Fudan FDP3P7

Support Compiler
Visual C++ 6.0
C++ Builder 6.0

Copyright

Copyright © 2004-2009 SMIMS Technology Corp. All rights reserved. No part of this publication may be reproduced or used in any form or by any means including graphic, electronic, or mechanical, or by photocopying, recording, taping, or information storage and retrieval systems, without written permission of SMIMS Technology Corp.

Trademarks

SMIMS is registered trademark of SMIMS Technology Corp. The VeriComm, VeriLite and VeriEnterprise are trademarks of SMIMS Technology Corp. All other products or services mentioned herein are trademarks of their respective holders and should be treated as such.

Table of Contents

1.	Introduction	1
1.1	General Features.....	1
1.2	Technical Support.....	2
2.	VeriSDK APIs	3
3.	Hardware interface	5
3.1	Pin Description.....	5
3.2	Timing Diagram	6
4.	Example – Inverter	8
4.1	Software Usage	8
4.2	Hardware Description	11
4.3	Timing diagram	11
5.	ezIF control Interface	13
5.1	ezIF interface introduction	13
5.2	Hardware Interface.....	14
5.3	Timing Diagram	15
5.4	Software Usage	17
6.	Pin Table.....	19
6.1	VeriSDK Pin Table	19
6.2	Other IOs Pin Table	20

List of Figures

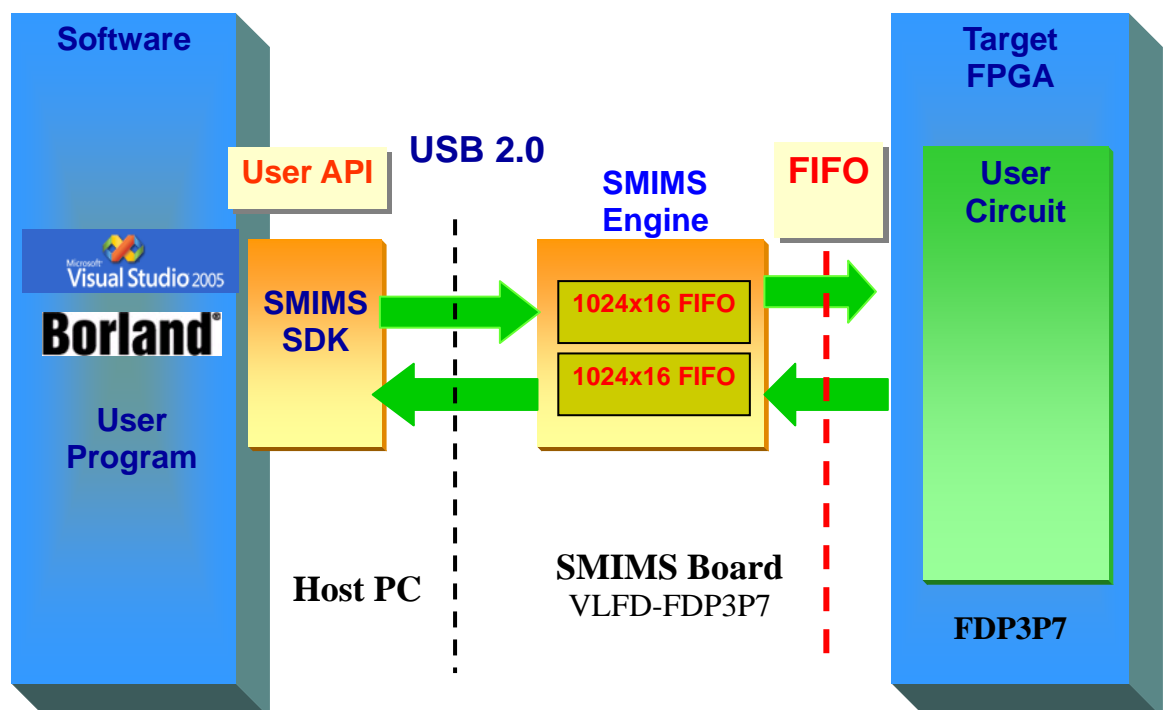
Figure 1 Timing Diagram of Single Read Data from SMIMS Engine FIFO buffer	6
Figure 2 Timing Diagram of Burst Read Data from SMIMS Engine FIFO buffer.....	7
Figure 3 Timing Diagram of Single Write Data from SMIMS Engine FIFO buffer.....	7
Figure 4 Timing Diagram of Burst Write Data to SMIMS Engine FIFO buffer.....	7
Figure 5 The state diagram of inverter example.....	11
Figure 6 The timing diagram of inverter example	12
Figure 7 Block diagram of ezIF control interface	13
Figure 8 IO Signals in the ezIF control interface	14
Figure 9 Timing diagram of writing a single data to memory	15
Figure 10 Timing diagram of writing burst data to memory.....	15
Figure 11 Timing diagram of read single data from Memory	16
Figure 12 Timing diagram of read burst data from Memory	16
Figure 13 Timing diagram of user design execution (the transceiver data count is different).....	16
Figure 14 Timing diagram of user design execution(the transceiver data count is the same)	17

List of Tables

Table 1. SMIMS VeriSDK APIs	4
Table 2. VeriSDK FIFO Interface signals and descriptions	6
Table 3. Inverter directory list in SDK/example/Inverter	8
Table 4. Signal description of ezIF hardware interface.....	15
Table 5. Pin table of VeriSDK for VLFD (FDP3P7) platform	19
Table 6. Pin Table of VLFD GIO	21

1. Introduction

The SMIMS provides the VeriSDK which is a C/C++/SystemC development APIs (SDK) to develop software applications for the interfacing and control of FPGA-based hardware applications, using these powerful and widely used software languages. Combining with the SMIMS hardware board and other software tools, the solution allows quick and easy hardware/software co-design and co-verification by the VeriSDK API interface.



The above block diagram is the VeriSDK architecture. There are two FIFO buffers in the SMIMS engine for Hardware-software communication.

1.1 General Features

- Support 32 bits windows-based platform
- VeriSDK C/C++ development API
- Support Microsoft Visual Studio 6.0 Visual C++

- Support Borland C++ Builder 6.0
- FPGA Configuration and control by USB through VeriSDK API
- Support up to 2 SMIMS motherboards in a single system

1.2 Technical Support

Please contact our technical support through one of the methods below.

- Telephone

Call our support hotline at 06-2381238 from 9:00 am to 6:00 pm (Taipei time), Monday to Friday.

- Instant Messaging

For real-time technical assistance, contact us through MSN. Our support account name is service@smims.com. Official support hours are from 9:00 am to 6:00 pm (Taipei time), Monday to Friday.

- E-mail

You may also send any queries to our email address service@smims.com

2. VeriSDK APIs

SMIMS VeriSDK provides the SDK APIs in the format of Dynamic Link Library (DLL). Though those APIs, the user C/C++ program may access/program FPGA data to achieve the hardware-software co-verification. Since the VeriSDK supports up to 2 SMIMS motherboards in a single system, the 1st parameter of those APIs indicates the selected board to operate. VeriSDK APIs are described in the following tables.

Function Name	Function Description
bool VLFD_AppOpen (int iBoard, char *Serial)	Initial the SMIMS Engine. Return true if success, false vice versa. The first input parameter is the board number. The others is the serial number.
bool VLFD_AppClose(int iBoard)	Close the SMIMS Engine, Return true if success, false vice versa. Input parameter is board number.
Bool VLFD_AppFIFOReadData (int iBoard, WORD *Buffer, unsigned size)	Read Data From Hardware (buffer of SMIMS Engine). The first parameter is board number. The 2 nd one is buffer to store data. The 3 rd one is total data count to read. If the operation is successful, this function will return true, false vice versa.
bool VLFD_AppFIFOWriteData (int iBoard, WORD *Buffer, unsigned size)	Send Data to Hardware (buffer of SMIMS Engine). The first parameter is board number. The 2 nd one is data buffer to send. The 3 rd one is total data count to Send. If the operation is successful, this function will return true, false vice versa.
bool VLFD_AppChannelSelector (int iBoard, BYTE channel)	Set 8 bits user-defined data. Note: If the function is called without FIFO data empty, this function will be block and APP_CH[7:0] will not changed immediately. APP_CH[7:0] will be altered after all data in FIFO buffer is

	read out.
bool VLFD_ProgramFPGA (int iBoard, char * BitFile)	Program FPGA. The first parameter is board number. The 2 nd one is the FPGA programming binary file.
Char* VLFD_GetLastErrorMsg(int iBoard)	Retrieve SMIMS Engine Error Message, if an error occurs.

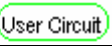
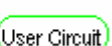
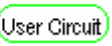
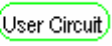
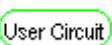
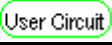
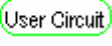
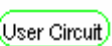
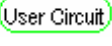
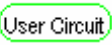
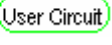
Table 1. SMIMS VeriSDK APIs

3. Hardware interface

The hardware interface between SMIMS engine and user's design in FPGA is used the FIFO interface. The interface's operation speed is based on the APP_CLK signal.

3.1 Pin Description

The signals and descriptions of hardware interface are listed in table 1.

Signal Name	Signal Type	Signal Description
APP_CLK	In → 	30 MHz Clock Source for VLFD platform
APP_RSTN	In → 	After programming a circuit into FPGA, a 1-to-0-to-1 pulse will be invoked to reset user's circuit.
APP_CS	In → 	When VLFD_AppOpen is called, the APP_CS will become 1. And, When VLFD_AppClose is called, the APP_CS will become 0.
APP_CH[7:0]	In → 	User defined 8-bits bus, Note when APP_CH changes the value, the data in the FIFO will be removed
APP_RD	← Out 	User design invokes this signal to read FIFO data from SMIMS engine.
APP_DI[15:0]	In → 	Data bus from SMIMS Engine FIFO.
APP_Empty	In → 	When this signal is 1, SMIMS Engine FIFO buffer for user design is empty. There is no data to be read.
APP_AlmostEmpty	In → 	When this signal is 1, SMIMS Engine FIFO buffer for user design is only one data available.
APP_WR	← Out 	User design invokes this signal to write data to SMIMS engine FIFO buffer.
APP_DO[15:0]	← Out 	Data bus from user design to be written to SMIMS Engine FIFO buffer.
APP_Full	In → 	When this signal is 1, SMIMS Engine FIFO buffer for storing user design's output data is full and it can not be written any data more.

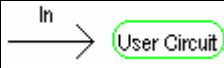
APP_AlmostFull		When this signal is 1, SMIMS Engine FIFO buffer for storing user design's output data is only one space left.
-----------------------	---	---

Table 2. VeriSDK FIFO Interface signals and descriptions

3.2 Timing Diagram

The FIFO Interface supports both single and burst read/write data. Before read/write data, the user should check the buffer statuses by those indicator signals of APP_Empty, APP_AlmostEmpty, APP_Full and APP_AlmostFull. The detail timing diagrams are shown as follows.

The timing diagram of single read data is shown as Fig. xxx. When APP_CS is high, APP_Empty is low and APP_RD activates for one clock period, the data will be read in the next clock period.

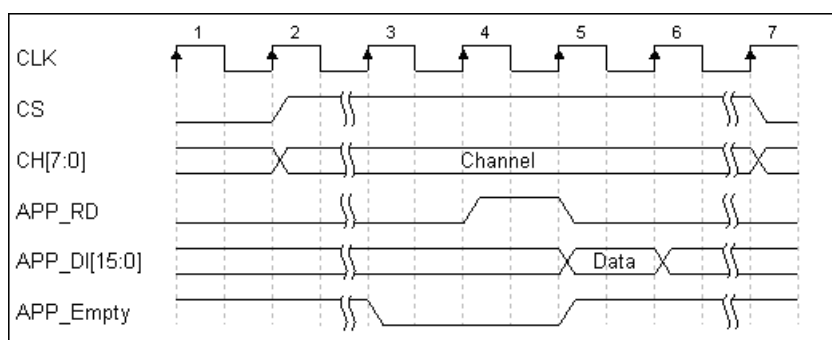


Figure 1 Timing Diagram of Single Read Data from SMIMS Engine FIFO buffer

The timing diagram of burst read data mode is shown as Fig 2. To read N data, check the APP_CS is high and APP_Empty/APP_AlomostEmpty is low, invoke the APP_RD for N clock period. The data will be in APP_DI which delay one clock cycles.

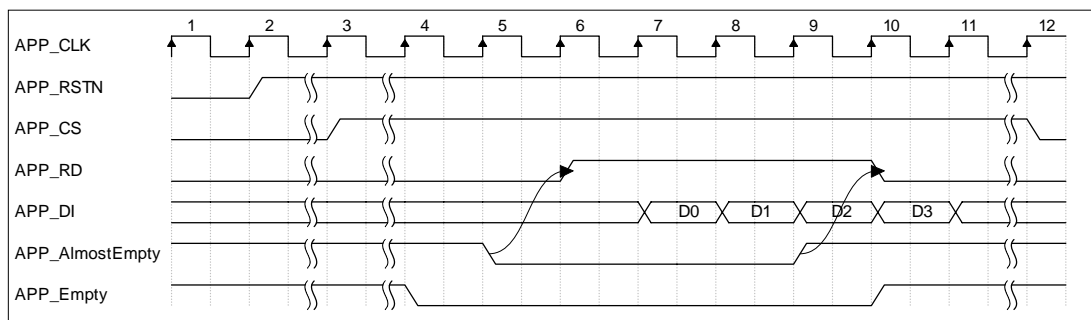


Figure 2 Timing Diagram of Burst Read Data from SMIMS Engine FIFO buffer

The timing diagram of single write data is shown as follows. When APP_CS is high, APP_Full is low. Send out the APP_WR and APP_DO data in the same time.

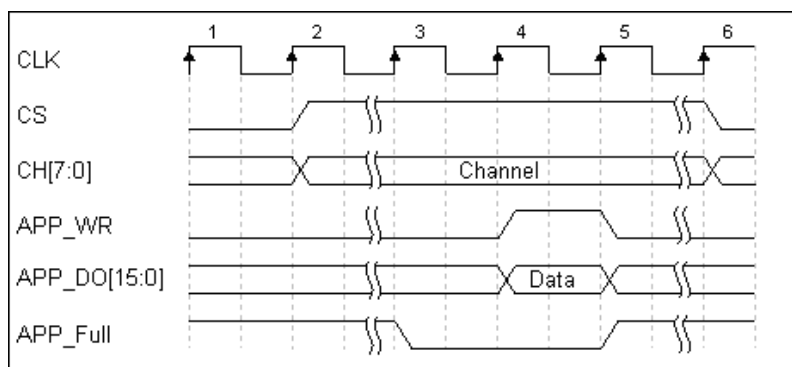


Figure 3 Timing Diagram of Single Write Data from SMIMS Engine FIFO buffer

The timing diagram of burst write data is shown as follows. When APP_CS is high, APP_Full/APP_AlmostFull are low. Send out the APP_WR and APP_DO data sequent in the same time.

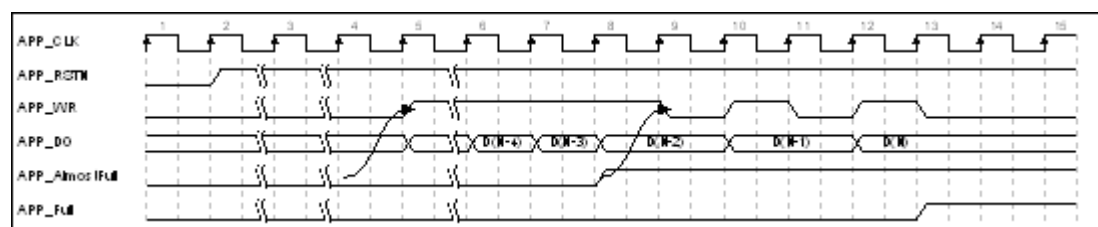
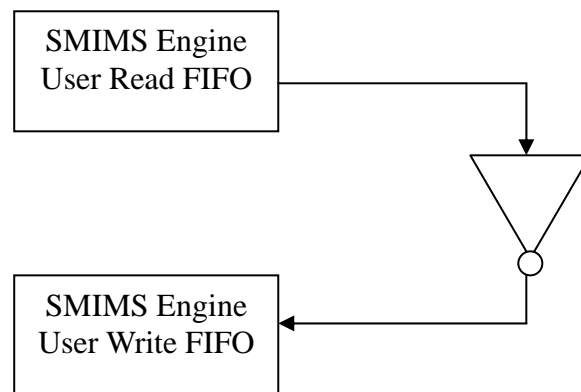


Figure 4 Timing Diagram of Burst Write Data to SMIMS Engine FIFO buffer

4. Example – Inverter

There are some examples in the SDK/example directory for your reference.

Here we will use an inverter as the example to show how to use the VeriSDK in Hardware/Software co-verification. In this example, we will send N numbers to FPGA and FPGA will invert those numbers and then send back to C program.



In this example, we provide the both Visual C++ and Borland C++ Builder example. The directory listed is shown as follows.

Directory	Description
bcbUseDll	Developed with Borland C++ Builder, utilizes Single-Thread
bcbUseDllMT	Developed with Borland C++ Builder, utilizes Multi-Thread
vcUseDll	Developed with Visual C++, utilizes Single-Thread
vcUseDllMT	Developed with Visual C++, utilizes Multi-Thread

Table 3. Inverter directory list in SDK/example/Inverter

4.1 Software Usage

The software is introduced using Borland C++ and Visual C++. This example utilizes the SMIMS APIs to provide a quick introduction on how to use VeriSDK.

Software program demonstration is shown as follows. (There are only partial codes described for demonstration purposes.

Please see the example for the complete version.)

```

=====
// Programming the FPGA
if( !VLFD_ProgramFPGA(NOW_USE_BOARD, "C:\\app_example.bit") )
{
    PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD) );
    return;
}

// Initialization of SMIMS VLFD V2
// The x's are filled in with the serial number found on the warranty card.
if ( !VLFD_AppOpen (NOW_USE_BOARD, "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx") )
{
    PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD) );
    return;
}
else
    PrintMsg("Initial successfully!");

// Prepare input patterns
for (int i=0;i<DataCount;i++)
    WriteBuffer[ i ] = i;

// Sending the data to the Hardware (FIFO)
bRet = VLFD_AppFIFOWriteData(NOW_USE_BOARD, WriteBuffer, DataCount);
if ( !bRet )
{
    PrintMsg("VLFD_AppFIFOWriteData failed!");
    PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
}

// For loop to transceiver data
for(int i=0; i<RunLoop - 1; i++)
{
    bRet = VLFD_AppFIFOWriteData(NOW_USE_BOARD, WriteBuffer, DataCount);
    if ( !bRet )
    {
        PrintMsg("VLFD_AppFIFOWriteData failed!");
        PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
    }
    bRet = VLFD_AppFIFOReadData(NOW_USE_BOARD, ReadBuffer, DataCount);
    if ( !bRet )
    {
        PrintMsg("VLFD_AppFIFOReadData failed!");
        PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
    }
}

// Receiving the data from Hardware (FIFO)
bRet = VLFD_AppFIFOReadData(NOW_USE_BOARD, ReadBuffer,DataCount);
if ( !bRet )
{
    PrintMsg("error: VLFD_AppFIFOReadData\n");
    PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
}

// Print out the results
for(int i=0;i<DataCount;i++)
{
    sprintf(cMsg,"%[d] %04X -> %04X",i, WriteBuffer[i], ReadBuffer[i]);

```

```
        PrintMsg(cMsg);
    }
    // Close SMIMS VLFD V2
    bRet = VLFD_AppClose(NOW_USE_BOARD);
    if ( !bRet )
    {
        PrintMsg("error: VLFD_AppClose\n");
        PrintMsg(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
        return;
    }
    else
    {
        PrintMsg("VLFD_AppClose OK!\n");
    }
}
```

Execution Results

```
=====
[0] 0000 -> FFFF
[1] 0001 -> FFFE
[2] 0002 -> FFFD
[3] 0003 -> FFFC
[4] 0004 -> FFFB
[5] 0005 -> FFFA
[6] 0006 -> FFF9
.... Etc
```

4.2 Hardware Description

The hardware design (single read/write) is to read data from Input FIFO, pass through an inverter, and then output the data to the Output FIFO. Here use a FSM (Finite state machine) to implement it. The state diagram is shown as follows.

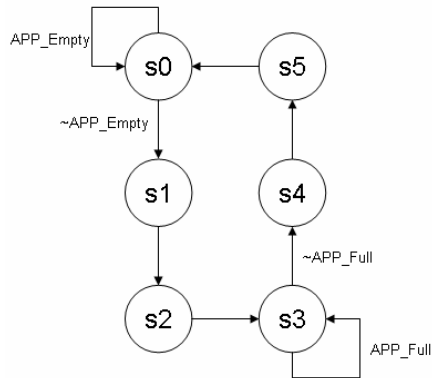


Figure 5 The state diagram of inverter example

The partial verilog code is shown as following.

```

=====
case(cur_state)
  s0 : begin // wait for input FIFO is not empty
    nxt_state = (APP_CS & ~APP_Empty) ? s1 : s0;
  end
  s1 : begin // start to read one data
    APP_RD = 1'b1;
    nxt_state = s2;
  end
  s2 : begin // latch input data
    LATCH_READ_DATA = 1'b1;
    nxt_state = s3;
  end
  s3 : begin // do inverter process and latch output data
    LATCH_WRITE_DATA = 1'b1;
    nxt_state = (APP_Full) ? s3 : s4;
  end
  s4 : begin // ready to write data to output FIFO
    APP_WR = 1'b1;
    nxt_state = s5;
  end
  s5 : begin // Back to state 0 and wait for another process
    nxt_state = s0;
  end
endcase

```

4.3 Timing diagram

The detailed timing diagram is shown in the following figure.

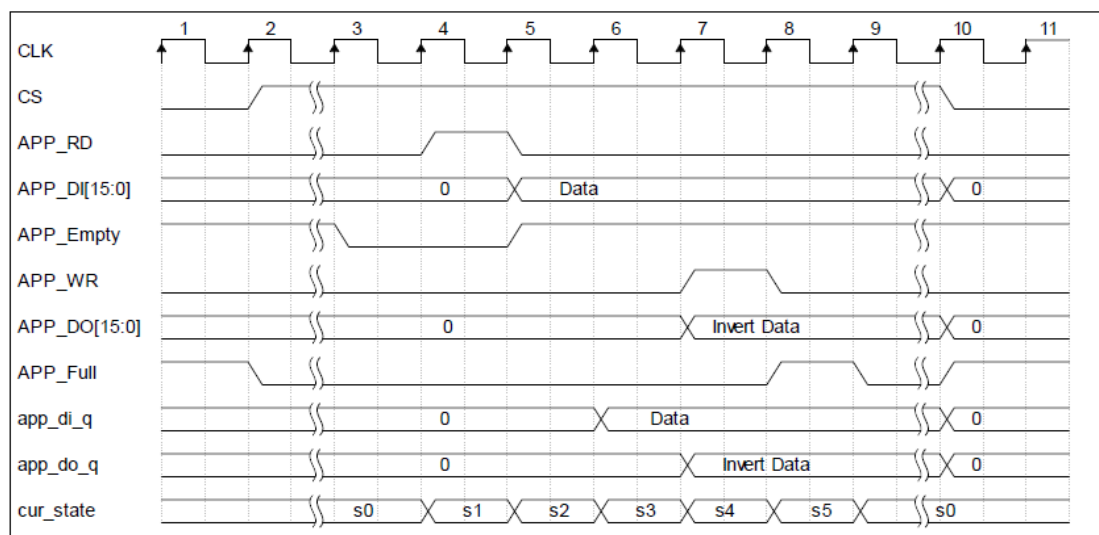


Figure 6 The timing diagram of inverter example

5. ezIF control Interface

Another example is ezIF control interface which is a data transmission management interface for handling FIFO data. It is a simple and easy-to-use wrapper to help user design quickly integrate to SMIMS platform.

ezIF is an interface which increases the efficiency and simplicity of incorporating the circuit into the SMIMS environment. The user may design their own FIFO communication interface instead of ezIF if an even more efficient operation method or versatile usage is required.

5.1 ezIF interface introduction

The ezIF assists the user regarding data transmission between the user design and the FIFO interface. The data transmitted from the software through this interface will be written into memory. When the data writing is complete, the user module will be signaled to execute. The user module will signal ezIF when the execution is complete. If a flag (finish signal) is raised, by the user module, ezIF will automatically return the data to the software. A block diagram of the above process can be shown as following figure.

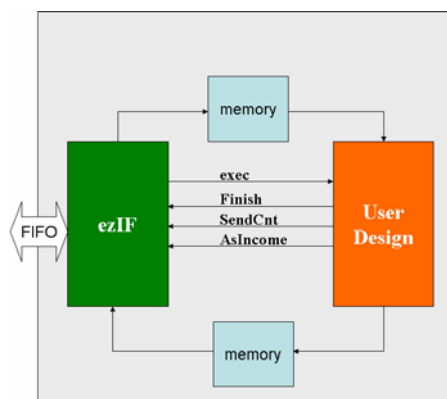


Figure 7 Block diagram of ezIF control interface

Its operation procedure is shown as follows

1. software set channel selector to 1 and send the data

- count(N). Then set channel selector to 0.
2. When FIFO is not empty, read 1st data from FIFO as the data count (N).
3. when FIFO is not empty, move N data to FPGA internal memory.
4. when step 2 is complete, invoke a exec signal to user design.
5. waiting for a finish signal and then disable the exec signal.
6. move M data (The number M is from SentCnt/AsIncome signal) from FPGA internal memory to FIFO.

5.2 Hardware Interface

The ezIF operation interface is shown as follows. The FIFO interface had been described in the chapter 3, we will only focus on the signals on the right side.

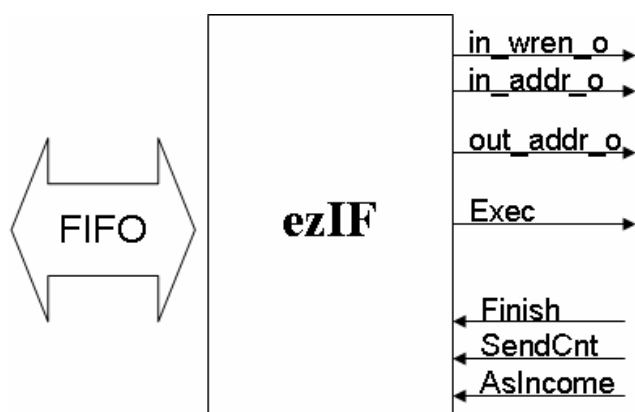


Figure 8 IO Signals in the ezIF control interface

The signals and descriptions of ezIF hardware interface are listed in the following table.

Name	Type	Descriptions
in_wren_o	Output	When the signal is 1, write the data into the memory.
in_addr_o	Output	Memory address for input.
out_addr_o	Output	Memory address for outputting data to the software.
Exec	Output	Execution signal for the circuit.

Finish	Input	Execution complete signal for the circuit.
SendCnt	Input	Number of data required to be returned to the software.
AsIncome	Input	If the number of returned data is equal to the number of inputted data, then this signal should be active.

Table 4. Signal description of ezIF hardware interface

5.3 Timing Diagram

ezIF writing a single string of data to memory (source of the data is APP_DI of FIFO)

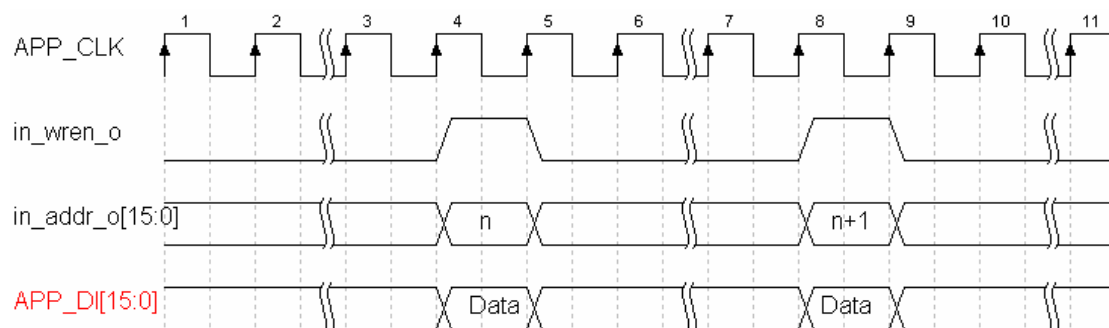


Figure 9 Timing diagram of writing a single data to memory

ezIF writing consecutive strings of data into memory (source of the data is APP_DI of FIFO)

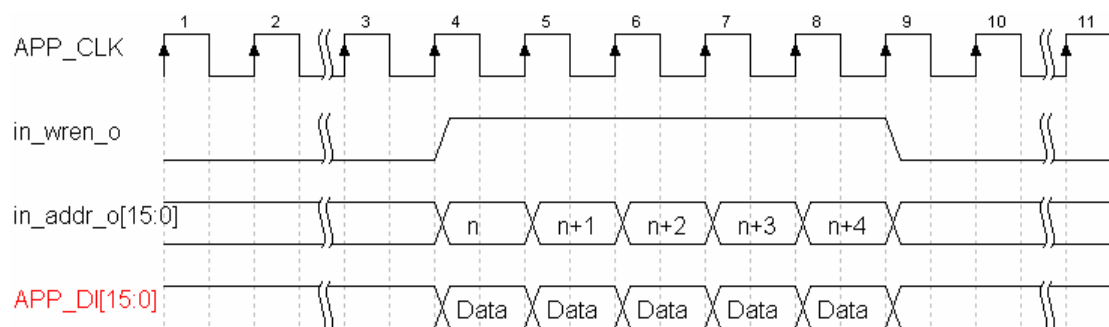


Figure 10 Timing diagram of writing burst data to memory

ezIF reads a single string of data from the memory and return it to software (Mem_Q is memory's output, user must connect it to APP_DO)

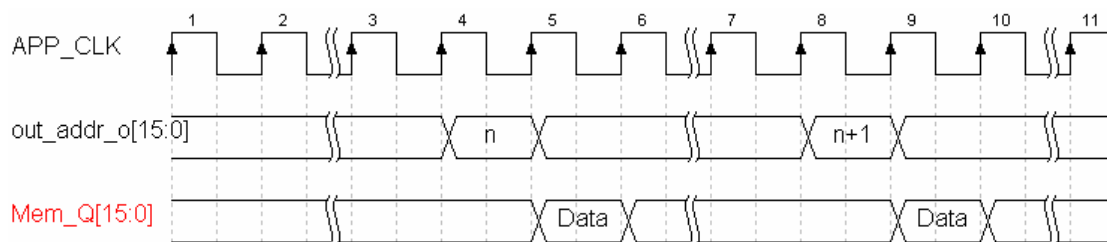


Figure 11 Timing diagram of read single data from Memory

ezIF reads consecutive strings of data from the memory and return it to software (Mem_Q is memory's output, user must connect it to APP_DO)

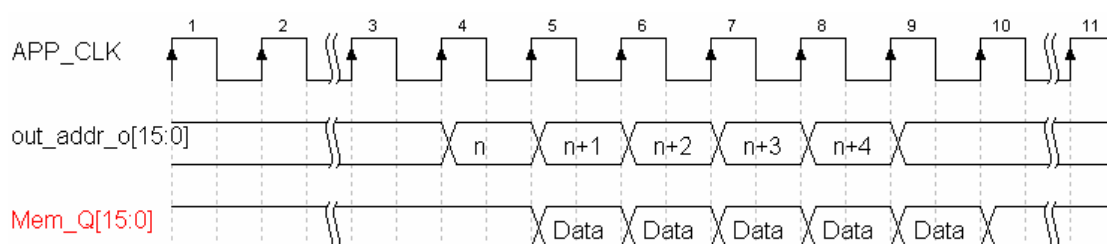


Figure 12 Timing diagram of read burst data from Memory

When ezIF completes the reception of data from software, it will raise and sustain a 1 at the Exec pin. The user can begin execution when that particular signal is received. When the execution ends, a Finish signal is sent and the Exec signal is then lowered. The number of data required to be returned to the software must be set when the finish signal is raised. ezIF will follow the setting and return the data back to the software.

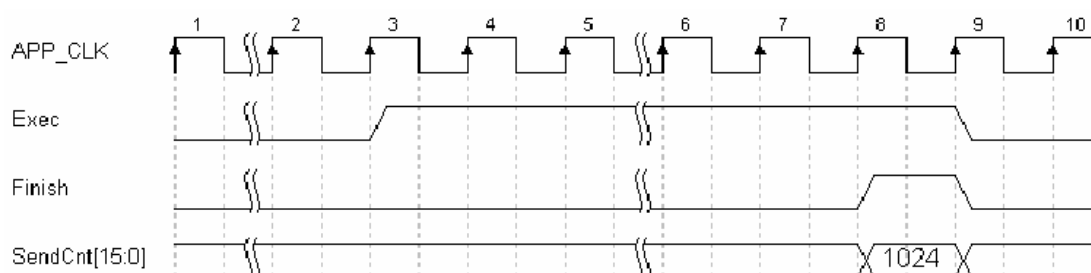


Figure 13 Timing diagram of user design execution (the transceiver data count is different)

The signal AsIncome is raised to 1 if the number of data returned is equal to the number of data inputted. At this time, the signal from SendCnt will be set to don't care.

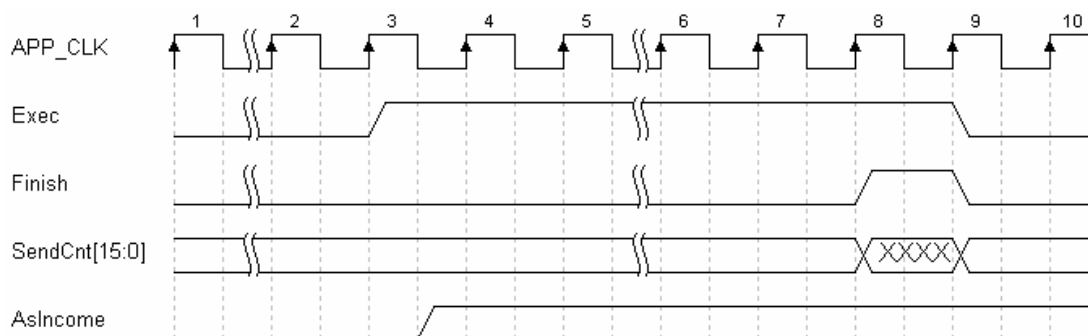


Figure 14 Timing diagram of user design execution(the transceiver data count is the same)

5.4 Software Usage

Before the data is sent, the user must first set the number of data required to be sent through software. During the configuration, the APP_CH signal of the FIFO must be set to 1. When the APP_CH is logic 1, the user can set number of data required to be sent. When the setting is complete, the APP_CH is set back to 0. The code of doing so is shown below.

```
VLFD_AppChannelSelector(NOW_USE_BOARD, 1)
Cmd = DataCount
VLFD_AppFIFOWriteData(NOW_USE_BOARD, &Cmd, 1);
VLFD_AppChannelSelector(NOW_USE_BOARD, 0)
```

Due to the long execution time of this configuration, it is suggested to only be done before every output if the number of data required to be sent are always different. Otherwise, it is suggested that the configuration is done once before multiple send/receive operations until the number of data required to be sent changes. At that time, the configuration will need to be redone.

Exampmary Code

```
DataCount = MAX_COUNT;
// set the number of data
if(!VLFD_AppChannelSelector(NOW_USE_BOARD, 1))
{
    printf(VLFD_GetLastErrorMsg(NOW_USE_BOARD) );
}
```

```

    VLFD_AppClose(NOW_USE_BOARD);
    return 1;
}
WriteBuffer[0] = DataCount;

bRet = VLFD_AppFIFOWriteData(NOW_USE_BOARD, WriteBuffer, 1);
if ( !bRet )
{
    printf("WriteData Cmd failed!");
    printf(VLFD_GetLastErrorMsg(NOW_USE_BOARD,));
}

if(!VLFD_AppChannelSelector(NOW_USE_BOARD, 0))
{
    printf(VLFD_GetLastErrorMsg(NOW_USE_BOARD) );
    VLFD_AppClose(NOW_USE_BOARD);
    return 1;
}

//Random integer generator
for(i=0;i<DataCount;i++)
    WriteBuffer[ i ] = rand() % 4096;

//Send data to FIFO (hardware)
bRet = VLFD_AppFIFOWriteData(NOW_USE_BOARD, WriteBuffer, DataCount);
if ( !bRet )
{
    printf("WriteData failed!");
    printf(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
}

//receive data from FIFO (hardware)
bRet = VLFD_AppFIFOReadData(NOW_USE_BOARD, ReadBuffer,
DataCount);
if ( !bRet )
{
    printf("error: VLFD_AppFIFOReadData\n");
    printf(VLFD_GetLastErrorMsg(NOW_USE_BOARD));
}
//check results
for(i=0;i<DataCount;i++)
{
    if(ReadBuffer[i] != 4096 - WriteBuffer[i])
    {
        printf("Error:%4d\n",ReadBuffer[i]);
    }
}

```

6. Pin Table

6.1 VeriSDK Pin Table

(FDP3P7)

訊號名稱	腳位編號		訊號名稱	腳位編號	
APP_CLK	P185	Input	CH[0]	P102	Input
APP_RSTN	P3	Input	CH[1]	P101	Input
APP_CS	P206	Input	CH[2]	P100	Input
APP_WR	P44	Output	CH[3]	P97	Input
APP_RD	P45	Output	CH[4]	P96	Input
APP_Full	P120	Input	CH[5]	P95	Input
APP_Empty	P115	Input	CH[6]	P89	Input
APP_AlmostFull	P116	Input	CH[7]	P88	Input
APP_AlmostEmpty	P114	Input			
APP_DI [0]	P151	Input	APP_DO [0]	P7	Output
APP_DI [1]	P148	Input	APP_DO [1]	P6	Output
APP_DI [2]	P150	Input	APP_DO [2]	P5	Output
APP_DI [3]	P152	Input	APP_DO [3]	P4	Output
APP_DI [4]	P160	Input	APP_DO [4]	P9	Output
APP_DI [5]	P161	Input	APP_DO [5]	P8	Output
APP_DI [6]	P162	Input	APP_DO [6]	P16	Output
APP_DI [7]	P163	Input	APP_DO [7]	P15	Output
APP_DI [8]	P164	Input	APP_DO [8]	P11	Output
APP_DI [9]	P165	Input	APP_DO [9]	P10	Output
APP_DI [10]	P166	Input	APP_DO [10]	P20	Output
APP_DI [11]	P169	Input	APP_DO [11]	P18	Output
APP_DI [12]	P173	Input	APP_DO [12]	P17	Output
APP_DI [13]	P174	Input	APP_DO [13]	P22	Output
APP_DI [14]	P175	Input	APP_DO [14]	P21	Output
APP_DI [15]	P191	Input	APP_DO [15]	P23	Output

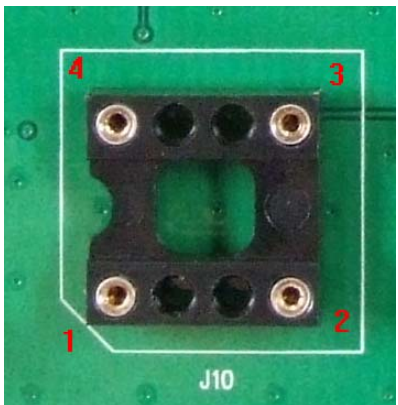
Table 5. Pin table of VeriSDK for VLFD (FDP3P7) platform

6.2 Other IOs Pin Table

LED (Low Active)		
PCB ID	FPGA Pin	Jumper Settings
D0	P147	J7 Short 1-2
D1	P145	J7 Short 3-4
D2	P141	J7 Short 5-6
D3	P140	J7 Short 7-8
D4	P139	J7 Short 9-10
D5	P138	J7 Short 13-14
D6	P136	J7 Short 15-16
D7	P135	J7 Short 17-18

Switch (Low Active)		
PCB ID	FPGA Pin	Jumper Settings
SW1.1	P134	J7 Short 19-20
SW1.2	P133	J7 Short 21-22

Button (Low Active)		
PCB ID	FPGA Pin	Jumper Settings
BT1	P132	J7 Short 23-24
BT2	P129	J7 Short 25-26

External Oscillator		
		
PCB ID	FPGA Pin	Jumper Settings
J10 – 1	NC	
J10 – 2	GND	
J10 – 3	P80	
J10 – 4	3.3V	

General I/O J7			
LED D0	1	2	P147
LED D1	3	4	P145
LED D2	5	6	P141
LED D3	7	8	P140
LED D4	9	10	P139
+5V	11	12	Direct connect J7.12 and J8.11
LED D5	13	14	P138
LED D6	15	16	P136
LED D7	17	18	P135
SW1.1	19	20	P134
SW1.2	21	22	P133
BT1	23	24	P132
BT2	25	26	P129
GND	27	28	P127
+3.3V	29	30	Direct connect J7.30 and J8.29
GND	31	32	P126
GND	33	34	P125
GND	35	36	P123
GND	37	38	P122
GND	39	40	P121

General I/O J8			
P147	1	2	P146
P145	3	4	N.C.
P141	5	6	N.C.
P140	7	8	N.C.
P139	9	10	N.C.
Direct connect J7.12 and J8.11	11	12	GND
P138	13	14	N.C.
P136	15	16	N.C.
P135	17	18	N.C.
P134	19	20	N.C.
P133	21	22	N.C.
P132	23	24	N.C.
P129	25	26	N.C.
P127	27	28	N.C.
Direct connect J7.30 and J8.29	29	30	GND
P126	31	32	N.C.
P125	33	34	N.C.
P123	35	36	N.C.
P122	37	38	N.C.
P121	39	40	N.C.

Table 6. Pin Table of VLFD GIO