'USB_IO_for_VB6.dll' User Guide

This guide explains the usage of 'USB_IO_for_VB6.dll' for creating custom gui's for DDC EVM board.

Note: This guide does not cover all the functions of the DLL but provides necessary information to configure the DDC EVM board and capture data from DDC device.

Overview:

The mode of communication between DDC EVM board and Computer is through USB. The DLL 'USB_IO_for_VB6.dll' used for communicating with the DDC EVM board creates necessary USB handle and closes it after use.

The DLL functions covered in this guide are,

1. XferINTDataIn - Receive data from USB to Computer.
2. XferINTDataOut - Send data from Computer to USB.
3. WriteFPGARegsC - Perform FPGA write operation and read back of FPGA Registers.
4. FastAllDataCap - Capture data from DDC device.

Function Declarations:

Note: The method for calling DLL function may vary with programming language/environment. The calling method for Visual Basic and LabVIEW are provided under each function explanation.

1. int __stdcall XferINTDataIn(int *USBdev, int *INTData, long *DataLength);
2. int __stdcall XferINTDataOut(int *USBdev, int *Data01, int *Data02, int *Data03, int *Data04);
3. long __stdcall WriteFPGARegsC(int *USBdev, long DUTSelect, long * RegsIn, long * RegsOut, long * RegEnable);
4. long __stdcall FastAllDataCap(double * AVGArr, double * RMSArr, double * P2PArr, double * MAXArr, double * MINArr, long ArrSize, long Channels, long nDVALIDReads, double * DataArray, long * AllDataAorBfirst);

Function Explanations:

*1. int __stdcall XferINTDataIn (int *USBdev, int *INTData, long *DataLength);*

Calling in Visual Basic:

Public Declare Function XferINTDataIn Lib "USB_IO_for_VB6.dll" (ByRef USBdev As Short, ByRef Array_Data As Short, ByRef DataLen As Integer) As Short

Calling in LabVIEW:

int16_t XferINTDataIn(int16_t *USBdev, int16_t *Array_Data, int32_t *DataLen);

Description:

Creates an instance of USB Device class. If the bulk in endpoint is reached, data is transferred and '0' is returned. Returns '-10' if USB endpoint cannot be reached. Returns '-2' if cannot open USB device. Returns '1' if more data is coming.

USBdev - Specify which DDC EVM board to receive data from, if multiple DDC EVM boards are connected to computer.

Array_Data - The variable to store the array of data received from DDC EVM board.

DataLen - Specify the length of Array_Data.

*2. int __stdcall XferINTDataOut(int *USBdev, int *RegH, int *RegL, int *DataH, int *DataL);*

Calling in Visual Basic:

Public Declare Function XferINTDataOut Lib "USB_IO_for_VB6.dll" (ByRef USBdev As Short, ByRef RegH As Short, ByRef RegL As Short, ByRef DataH As Short, ByRef DataL As Short) As Short

Calling in LabVIEW:

int16_t XferINTDataOut(int16_t *USBdev, int16_t *RegH, int16_t *RegL, int16_t *DataH, int16_t *DataL);

Description:

Creates an instance of USB Device class. If the bulk out endpoint is reached, data is transferred and '0' is returned. Returns '-1' if no USB devices are open.

USBdev - Specify which DDV EVM board to receive data from, if multiple DDV EVM boards are connected to computer.

RegH - The decimal equivalent of high nibble of 8 bits FPGA Register Address.

RegL - The decimal equivalent of low nibble of 8 bits FPGA Register Address.

DataH - The decimal equivalent of high nibble of 8 bits FPGA Register Value.

DataL - The decimal equivalent of low nibble of 8 bits FPGA Register Value.

*3. long __stdcall WriteFPGARegsC(int *USBdev, long DUTSelect, long *RegsIn, long *RegsOut, long *RegEnable);*

Calling in Visual Basic:

Public Declare Function WriteFPGARegsC Lib "USB_IO_for_VB6.dll" (ByRef USBdev As Short, ByVal DUTSelect As Integer, ByRef Array_RegsIn As Integer, ByRef Array_RegsOut As Integer, ByRef Array_RegEnable As Integer) As Integer

Calling in LabVIEW:

int32_t WriteFPGARegsC(int16_t *USBdev, int32_t *DUTSelect, int32_t *Array_RegsIn, int32_t *Array_RegsOut, int32_t *Array_RegEnable);

Description:

Creates an instance of USB Device class. Writes, reads back FPGA register content, then resets CONV. Returns '-1' if no USB devices are open. Returns '0' upon completion of function. Returns '-9' or '-10' if USB endpoint cannot be reached.

USBdev - Specify which DDV EVM board to receive data from, if multiple DDV EVM boards are connected to computer.

DUTSelect - (Currently unused).

Array_RegsIn - Array of Register Values in decimal format. The Value for the Register Address 'X' must be at the index 'X' in the Array_RegsIn. These Register Values will be written to the FPGA Registers.

Array_RegsOut - Array of Register Values in decimal format. The Value for the Register Address 'X' will be at the index 'X' in the Array_RegsOut. These Register Values are read from FPGA after performing write operation.

Array_RegsEnable - Array of values with either '0' or '1' indicating which FPGA Register are to be written. The Register with Address 'X' will be written only if the value of Array_RegsEnable at index 'X' is '1'.

*4. long __stdcall FastAllDataCap(double * AVGArr, double *RMSArr, double *P2PArr, double *MAXArr, double *MINArr, long ArrSize, long Channels, long Samples, double *AllData, long *AllDataAorBfirst);*

Calling in Visual Basic:

Public Declare Function FastAllDataCap Lib "USB_IO_for_VB6.dll" (ByRef AVGArr As Double, ByRef RMSArr As Double, ByRef P2PArr As Double, ByRef MAXArr As Double, ByRef MINArr As Double, ByVal ArrSize As Integer, ByVal Channels As Integer, ByVal Samples As Integer, ByRef AllData As Double, ByRef AllDataAorBfirst As Integer) As Integer

Calling in LabVIEW:

int32_t FastAllDataCap(double *AvgArr, double *RMSArr, double *P2PArr, double *MaxArr, double *MinArr, int32_t ArrSize, int32_t Channels, int32_t Samples, double *AllData, int32_t *AllDataAorBFirst);

Description:

Creates an instance of USB Device class. Captures data from DDC device and provides the samples read, the average per channel, the RMS value per channel, the peak to peak difference per channel, maximum value and minimum value per channel. Returns '0' if the data is captured properly. If the return value is less than '0', the data capture was not proper.

AvgArr - The average of samples read per channel.

RMSArr - The RMS value of samples read per channel.

P2Parr - The peak to peak difference of the read samples per channel.

MaxArr - The maximum sample value per channel.

MinArr - The minimum sample value per channel.

ArrSize - The array length of above mentioned arrays which must be equal to twice the number of channels to read.

Channels - The number of channels to read from DDC Device.

Samples - The number of samples to read from each channel.

AllData - The entire samples read.

AllDataAorBFirst - Specifies whether A side is read first or B side is read first.

Function Usage:

1. Configuring FPGA Registers:

The Registers of FPGA to be configured are mentioned below.

Register Address (hex)	Register Value
01	Load CONV Low MSB
02	Load CONV Low MIDB
03	Load CONV Low LSB
04	Load CONV High MSB
05	Load CONV High MIDB
06	Load CONV High LSB
07	DDC Sys Clock High and Low (each 4 bits)
08	DDC Sys Clock Select (0 - Low, 1 - Running)
09	Format*16 + Channel Count Format: 0 – 16 bit, 1 – 20 bit Ch. Count: 0 – 1 Ch., 1 – 2 Ch., 2 – 4 Ch., 3 – 8 Ch., 4 – 16 Ch., 5 – 32 Ch., 6 – 64 Ch., 7 – 128 Ch., 8 – 256 Ch., 9 – 512 Ch., 10 – 1024 Ch., 11 – 0 Ch.
0A	DDC Data Clock High and Low (each 4 bits)
0B	DDC Data Clock Select (0 - Low, 1 - Running)
0C	DVALIDS (Samples) to Ignore
0D	DVALIDS (Samples) to Read MSB
0E	DVALIDS (Samples) to Read MIDB
0F	DVALIDS (Samples) to Read LSB
13	DCLK Wait Count MSB
14	DCLK Wait Count MSB
20	CLK CFG High and Low (each 4 bits)

The high and low bytes FPGA Firmware version can be read from Register Addresses 0x5E and 0x5F respectively.

Use **WriteFPGARegsC()** for configuring FPGA Registers.

Initialize three arrays with 255 elements each, for RegsIn, RegsOut and RegsEnable. Assign values to be written in RegsIn at corresponding indices and '1' at the same indices in RegsEnable. Specify 0 for USBdev to configure first DDC EVM board.

After the DLL function execution, the readback FPGA register values will be assigned in RegsOut array at corresponding indices. Use these values to verify write operation.

Note: To perform FPGA registers read operation alone, make all the values in RegsEnable as '0' and execute the DLL function.

2. Configuring DDC Registers:

First use **XferINTDataOut()** to send data to board and configure DDC register.

Since the function at a time writes to only one register, the function has to be executed for multiple times to write to the below mentioned registers in FPGA. The first two digits represent RegH & RegL and the last two digits represent DataH & DataL.

Data to be sent in sequence: 0000, 0000, 1100, 1200, 1C<DDC_High_Reg Value>, 1D<DDC_Low_Reg Value>, 1F01, 0000, 0000, 1E01.

Then use **XferINTDataIn()** to receive data from board which gives the read back values from DDC device. This is the optional read back from the DDC device immediately after configuring the DDC registers.

The data length must be 512 and the array should be initialized with 4096 elements. After the DLL function execution, the array specified as a argument in DLL function contains read back data.

Parsing the read back data for DDC registers value and Die Id:

1. First retrieve elements from all the even indices.
2. Now delete the elements at 0, 4, 8, 12… indices
3. The elements at indices 1, 5, 9, 13… can be left as such
4. Then replace each elements at indices 2, 3, 6, 7, 10, 11… with two elements that are the decimal equivalent of high nibble and low nibble of the corresponding element respectively.

The array now contains DDC Register data and Die Id at specific indices depending on the DDC device present on the EVM board. Each element in this array represents either high or low nibble of the data.

Index	Device	Value
64	DDC264	High Register High Nibble (if 16 bit mode)
65	DDC264	High Register Low Nibble (if 16 bit mode)
66	DDC264	Low Register High Nibble (if 16 bit mode)
67	DDC264	Low Register Low Nibble (if 16 bit mode)
68	DDC264	Die Id (if 16 bit mode)
192	DDC264	High Register High Nibble (if 16 bit mode)
193	DDC264	High Register Low Nibble (if 16 bit mode)
194	DDC264	Low Register High Nibble (if 16 bit mode)

195	DDC264	Low Register Low Nibble (if 16 bit mode)
196	DDC264	Die Id (if 16 bit mode)
80	DDC264	High Register High Nibble (if 20 bit mode)
81	DDC264	High Register Low Nibble (if 20 bit mode)
82	DDC264	Low Register High Nibble (if 20 bit mode)
83	DDC264	Low Register Low Nibble (if 20 bit mode)
84	DDC264	Die Id (if 20 bit mode)
240	DDC264	High Register High Nibble (if 20 bit mode)
241	DDC264	High Register Low Nibble (if 20 bit mode)
242	DDC264	Low Register High Nibble (if 20 bit mode)
243	DDC264	Low Register Low Nibble (if 20 bit mode)
244	DDC264	Die Id (if 20 bit mode)

Data at indices 64,65,66,67,68 should match data at indices 192,193,194,195,196 for 16 bit mode and data at indices 80,81,82,83,84 should match data at indices 240,241,242,243,244 for 20 bit mode if the DDC is configured successfully.

To **Hard Reset** the DDD device use **XferINTDataOut()** to send the following register values. The first two digits represent RegH & RegL and the last two digits represent DataH & DataL.

Data to be sent in sequence: 0000, 15FF, 15FF, 1500, 1500, 15FF, 15FF.

3. Data Capture:

Use **FastAllDataCap()** for Data Capture. Before data capture, FPGA and DDC must be configured properly.

Initialize the arrays AVGArr, RMSArr, P2Parr, MAXArr, MINArr with Channels*2 elements.

Initialize the array AllData with Channels*Samples elements.

After the DLL function execution, the arrays will be filled with corresponding data.

AllData array will have data in the order as shown below.

(**X** – Channels Count, **N** – Samples Count)

AllData Array Index	Side		Sample #	Channel #
	AllDataAorBFirst = 0	AllDataAorBFirst = 1		
0	A	B	1	1
1	A	B	1	2
…	…	…	…	…
(X – 1)	A	B	1	X
X	B	A	1	1
(X + 1)	B	A	1	2
…	…	…	…	…
(2X – 1)	B	A	1	X
2X	A	B	2	1
(2X + 1)	A	B	2	2
…	…	…	…	…
(3X – 1)	A	B	2	X
3X	B	A	2	1
(3X + 1)	B	A	2	2

…	…	…	…	…
(4X – 1)	B	A	2	X
……	……	……	……	……
X*(N-2)	A	B	N	1
(X*(N-2) + 1)	A	B	N	2
…	…	…	…	…
(X*(N-1) – 1)	A	B	N	X
X*(N-1)	B	A	N	1
(X*(N-1) + 1)	B	A	N	2
…	…	…	…	…
(X*N – 1)	B	A	N	X

The AVGArr, RMSArr, P2Parr, MAXArr and MINArr arrays will have corresponding data in the order A_ChX, A_Ch(X-1), … , A_Ch1, B_ChX, B_Ch(X-1),…, B_Ch1. These arrays will always have A side's data first and B side's data second and from last channel to first channel.