Protouch2 USB57X4 Linux SDK User Guide

Version 1.0

Table of Contents

1 Introduction

Protouch2 Linux SDK allows the user to access the USB57X4 family of Microchip hubs. It can also be used for exercising unique features like FlexConnect, configuration programming, Register read/write, GPIO, SPI and I2C bridging on top of the hub functionality.

2 Legal Information

3 Package Content

libpt2: This Folder contains source code for building static library

- Makefile
- README
- MchpUSBInterface.cpp
- MchpUSBInterface.h
- typedef.h
- USB2740_SpiFlash.cpp
- USB2740_SpiFlash.h
- USBHubAbstraction.cpp
- USBHubAbstraction.h

examples: This Folder contains sample code for following features

- **Flexconnect**
  - FlexConnect.cpp
  - Makefile
  - README

- **gpio**
  - gpio.cpp
  - Makefile
  - README

- **I2C_Bridging**
  - i2c_bridging.cpp
  - Makefile
  - README

- **OTPProgrammer**
  - OTP_Programmer.cpp
  - Makefile
  - README

- **register_rw**
  - register_rw.cpp
  - Makefile
  - README

- **SPI_Bridging**
  - spi_bridging.cpp
  - Makefile
  - README

4 List of APIs

4.1 Device Open / Close APIs

Functions

	Name	Description
◆	MchpUsbOpenID	Open the device handle.
◆	MchpUsbClose	Close the device handle.

Description

This section covers APIs which enable open / close of the device. Before issuing any command to the device, the handle needs to be opened first.

4.1.1 MchpUsbOpenID Function

C

```
HANDLE MchpUsbOpenID(
    UINT16 wVID,
    UINT16 wPID
);
```

Description

This API will return handle to the first instance of the HUB VID & PID matched device.

Preconditions

- None.

Parameters

Parameters	Description
wVID	Vendor ID(VID) of the Hub.
wPID	Product ID(PID) of the Hub.

Returns

HANDLE of the Vendor ID and Product ID matched hub - for success

INVALID_HANDLE_VALUE (Call GetMchpUsbLastErr for more details) - for failure

Remarks

None

Example

```
CHAR sztext[2048];

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");
```

4.1.2 MchpUsbClose Function

C

```
BOOL MchpUsbClose(
    HANDLE DevID
);
```

Description

This API will close the handle for device specified in the call.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device - Return value of MchpUsbOpenID.

Returns

TRUE - for Success;

FALSE - for Failure

Remarks

None

Example

```
hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

if (FALSE == MchpUsbClose(hDevice))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04x",dwError);
    exit (1);
}
```

4

4.2 GPIO Bridging APIs

Functions

	Name	Description
	MchpUsbConfigureGPIO	This API configures the specified PIO line for general purpose input/output (GPIO)
	MchpUsbGpioGet	Get the state of the specified GPIO pin
	MchpUsbGpioSet	Set the state of the specified GPIO pin

Description

This APIs are used for low level control of GPIO pins in Microchip USB hubs. User can configure the direction, pull up / down, read data & write data to any GPIO.

4.2.1 MchpUsbConfigureGPIO Function

C

```
BOOL MchpUsbConfigureGPIO(
    HANDLE DevID,
    UINT8 PIONumber
);
```

Description

This API configures the specified PIO line for general purpose input/output (GPIO).

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
PIONumber	The GPIO pin number to be configured as GPIO mode.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");
//Configure pin number 11 as GPIO

if (FALSE ==MchpUsbConfigureGPIO(hDevice,11))
{

        dwError = MchpUsbGetLastErr(hDevice);

        printf ("Failed to open the device Error,%04x",dwError);

        exit (1);
}
```

4.2.2 MchpUsbGpioGet Function

C

```
BOOL MchpUsbGpioGet(
    HANDLE DevID,
    UINT8 PIONumber,
    UINT8* Pinstate
);
```

Description

This API gets the state of the specified GPIO pin. The direction of the GPIO pin referred in PIONumber is set to IN in this function.

Preconditions

MchpUsbOpenID should be called before calling this API

PIN should be configured as GPIO mode before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
PIONumber	The GPIO pin number from which to read the pin state
Pinstate	1 = Pin state is High 0 = Pin state is Low

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```
hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfully");

//Get status of pin number gpio 11
byData = 0x00;
```

```
//Configure pin number 11 as GPIO

if (FALSE ==MchpUsbConfigureGPIO(hDevice,11))
{

        dwError = MchpUsbGetLastErr(hDevice);

        printf ("Failed to open the device Error,%04x",dwError);

        exit (1);
}

//GPIO set

int PIONumber=11;


int PINState=0;

if (FALSE == MchpUsbGpioGet (hDevice,PIONumber,&PINState))

{

    dwError = MchpUsbGetLastErr(hDevice);
    exit (1);

}
```

4.2.3 MchpUsbGpioSet Function

C

```
BOOL MchpUsbGpioSet(
    HANDLE DevID,
    UINT8 PIONumber,
    UINT8 Pinstate
);
```

Description

This API sets the state of the specified GPIO pin with the state mentioned in Pinstate. The GPIO pin direction is set to OUT in this function.

Preconditions

MchpUsbOpenID should be called before calling this API

PIN should be configured as GPIO mode before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
PIONumber	The GPIO pin number from which to write the pin state
Pinstate	1 = Pin state is High 0 = Pin state is Low

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```
hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfully");

//Toggle PIONumber 11

//Configure pin number 11 as GPIO

if (FALSE ==MchpUsbConfigureGPIO(hDevice,11))
{

        dwError = bMchpUsbGetLastErr(hDevice);

        printf ("Failed to open the device Error,%04x",dwError);

        exit (1);
}

//GPIO set

int PIONumber=11;

int PINState=1;

if (FALSE == MchpUsbGpioSet (hDevice,PIONumber,PINState))

{

    dwError = MchpUsbGetLastErr(hDevice);

    exit (1);

}
```

4.3 XDATA Bridging APIs

Functions

	Name	Description
◆	MchpUsbRegisterRead	Read the XDATA register(s) in the XDATA space of of internal registers.
◆	MchpUsbRegisterWrite	Write to the XDATA register (s) in the XDATA space of the internal registers.

Description

This section lists the APIs that enable read / write of register space in Microchip USB hubs.

4.3.1 MchpUsbRegisterRead Function

C

```
BOOL MchpUsbRegisterRead(
    HANDLE DevID,
    UINT16 RegisterAddress,
    UINT16 BytesToRead,
    UINT8* InputData
);
```

Description

This API for Read the XDATA register(s) in the XDATA space of internal registers. This API also does the following: 1) Checks for the correct device handle before reading the registers 2) Checks for the proper buffer length

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device. Before calling this API ,the caller must acquire the device handle by calling appropriate API.
RegisterAddress	Start Address(in the XDATA space) from where Read operation starts
BytesToRead	Number of bytes to be read
InputData	Pointer to the buffer where data from XDATA registers will be stored.Caller must allocate memory for the buffer to accommodate the number of bytes to be read.

Returns

TRUE - for Success; FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

None

Example

```
hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfully");
```

```
UINT8 byData = 0x55;
UINT16 wAddr = 0x4800;
printf("Xdata Write operation, Write 0x%02x in 0x%04xn",byData,wAddr);
if (FALSE ==MchpUsbRegisterWrite(hDevice,wAddr,1,&byData))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Failed to open the device Error,%04x",dwError);
    exit (1);
}
cout << "Success :Xdata Write operation";
byData = 0x00;

cout << "Xdata Read operation";
if (FALSE ==MchpUsbRegisterRead(hDevice,wAddr,1,&byData))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
cout << "Success : Xdata Read operation";
printf("Xdata Read value is %02x from 0x%04x n",byData,wAddr);
```

4.3.2 MchpUsbRegisterWrite Function

C

```
BOOL MchpUsbRegisterWrite(
    HANDLE DevID,
    UINT16 RegisterAddress,
    UINT16 BytesToWrite,
    UINT8* OutputData
);
```

Description

This API for Write to the XDATA register(s) in the internal registers. This API also does the following: 1) Checks for the correct device handle before reading the registers 2) Checks for the proper buffer length

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device. Before calling this API ,the caller must acquire the device handle by calling appropriate API.
RegisterAddress	Start Address(in the XDATA space) from where Write operation starts
BytesToWrite	Number of bytes to be write
OutputData	Pointer to the buffer containing data to write to XDATA registers.

Returns

TRUE - for Success; FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

```
None
```

Example

```
hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfully");
```

```
UINT8 byData = 0x55;
UINT16 wAddr = 0x4800;
printf("Xdata Write operation, Write 0x%02x in 0x%04xn",byData,wAddr);
if (FALSE ==MchpUsbRegisterWrite(hDevice,wAddr,1,&byData))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Failed to open the device Error,%04x",dwError);
    exit (1);
}
cout << "Success :Xdata Write operation";
byData = 0x00;

cout << "Xdata Read operation";
if (FALSE ==MchpUsbRegisterRead(hDevice,wAddr,1,&byData))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
cout << "Success : Xdata Read operation";
printf("Xdata Read value is %02x from 0x%04x n",byData,wAddr);
```

4.4 I2C Bridging APIs

Functions

	Name	Description
◆	MchpUsbI2CSetConfig	Set I2C config parameters
◆	MchpUsbI2CRead	I2C read through the I2C pass-through interface of USB device
◆	MchpUsbI2CWrite	I2C write through the I2C pass-through interface of USB device
◆	MchpUsbI2CTransfer	I2C read and write through the I2C pass-through interface of USB device

Description

Microchip USB hubs facilitate USB-I2C bridging through USB control point of the embedded USB device (5th port).

4.4.1 MchpUsbI2CSetConfig Function

C

```
BOOL MchpUsbI2CSetConfig(
    HANDLE DevID,
    INT CRValue,
    INT nValue
);
```

Description

This function enables I2C pass-through and the clock rate of the I2C Master device.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
CRValue	Clock Rate value of the I2C clock if nValue is zero.
nValue	1 = 62.5Khz 2 = 235KHz 3 = 268KHz 4 = 312kHz 5 = 375KHz Other values are Reserved. CRValue is dont care if nValue is nonzero.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;
```

```
hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

INT iClockRate = 1; //62.5 KHz

//To Read EEPROM AT24C04
//Set desired value in clock
if(FALSE == MchpUsbI2CSetConfig(hDevice,0,iClockRate))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Error: MchpUsbI2CSetConfig- %04xn",(unsigned int)dwError);
    exit (1);
}
```

4.4.2 MchpUsbI2CRead Function

C

```
BOOL MchpUsbI2CRead(
    HANDLE DevID,
    INT BytesToRead,
    UINT8* InputData,
    UINT8 SlaveAddress
);
```

Description

This API performs an I2C read through the I2C pass-through interface of USB device.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
BytesToRead	Number of bytes to be read. Maximum value can be 512.
InputData	Pointer to the Buffer containing I2C read data
SlaveAddress	I2C Slave address

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

INT iClockRate = 1; //62.5 KHz

//To Read EEPROM AT24C04
//Set desired value in clock
if(FALSE == MchpUsbI2CSetConfig(hDevice,0,iClockRate))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Error: MchpUsbI2CSetConfig- %04xn",(unsigned int)dwError);
    exit (1);
}
 //Read 512 bytes
UINT8 byReadData[512];
if(FALSE == MchpUsbI2CRead(hDevice,512,&byReadData[0],0x50) )
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Failed to Read- %04xn",(unsigned int)dwError);
    exit (1);
}
```

4.4.3 MchpUsbI2CWrite Function

C

```
BOOL MchpUsbI2CWrite(
    HANDLE DevID,
    INT BytesToWrite,
    UINT8* OutputData,
    UINT8 SlaveAddress
);
```

Description

This API performs an I2C write through the I2C pass-through interface of USB device.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
BytesToWrite	Number of bytes to be write. Maximum value can be 512.
OutputData	Pointer to the Buffer containing I2C data to be written. Cannot be a constant value
SlaveAddress	I2C Slave address

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;
```

```
UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

INT iClockRate = 1; //62.5 KHz

//To Read EEPROM AT24C04
//Set desired value in clock
if(FALSE == MchpUsbI2CSetConfig(hDevice,0,iClockRate))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Error: MchpUsbI2CSetConfig- %04xn",(unsigned int)dwError);
    exit (1);
}
UINT8 byWriteData[9];

//Set start address
byWriteData[0] = 0x00;

if(FALSE== MchpUsbI2CWrite(hDevice,9,(BYTE *)&byWriteData,0x50))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Failed to write- %04xn",(unsigned int)dwError);
    exit (1);
}
```

4.4.4 MchpUsbI2CTransfer Function

C

```
BOOL MchpUsbI2CTransfer(
    HANDLE DevID,
    BOOL bDirection,
    UINT8* pbyBuffer,
    UINT16 wDataLen,
    UINT8 bySlaveAddress,
    BOOL bStart,
    BOOL bStop,
    BOOL bNack
);
```

Description

This API performs an I2C read and write through the I2C pass-through interface of USB device.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
bDirection	0 : I2C Write 1 : I2C Read

pbyBuffer	I2C Write - Pointer to the buffer which contains the data to be sent over I2C I2C Read - Pointer to the buffer to which the data read from I2C will be stored. Cannot be a constant value
DataLength	I2C Write - Number of bytes to write I2C Read - Number of bytes to read Maximum value can be 512 .
bySlaveAddress	Slave address of the I2C device
bStart	Indicates whether the start condition needs to be generated for this transfer, useful when combining single transfer in multiple API calls to handle large data. TRUE (Generates Start condition) FALSE(Does not generate Start condition)
bStop	Indicates whether the stop condition needs to be generated for this transfer, useful when combining single transfer in multiple API calls to handle large data. TRUE (Generates Stop condition) FALSE(Does not generate Stop condition)
bNack	Indicates whether the last byte should be NACK'ed for this transfer. TRUE (Generates NACK condition for the last byte of the transfer) FALSE(Does not generate NACK condition)

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

INT iClockRate = 1; //62.5 KHz

//To Read EEPROM AT24C04
//Set desired value in clock
if(FALSE == MchpUsbI2CSetConfig(hDevice,0,iClockRate))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Error: MchpUsbI2CSetConfig- %04xn",(unsigned int)dwError);
    exit (1);
}
//For i2c eeprom at24c04 ,read 10 bytes
UINT8 byData[512];
UINT8 byBytetoWrite = 0x00; //Write address first
if(FALSE == MchpUsbI2CTransfer(hDevice,0,byBytetoWrite,1,0x50,1,1,0))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("Error: I2C Transfer Failed- %04xn",(unsigned int)dwError);
    exit (1);
}
 //Read 10 bytes
if(FALSE == MchpUsbI2CTransfer(hDevice,1,byData[0],10,0x50,1,1,1))
{
    dwError = MchpUsbGetLastErr(hDevice);
```

```
     printf("Error: I2C Transfer Failed- %04xn",(unsigned int)dwError);
     exit (1);
 }
```

4.5 SPI Bridging APIs

Functions

	Name	Description
◆	MchpUsbSpiSetConfig	This API enables/disables the SPI interface.
◆	MchpUsbSpiFlashRead	This API performs read operation from SPI Flash.
◆	MchpUsbSpiFlashWrite	This API performs write opeartion to SPI Flash memory.
◆	MchpUsbSpiTransfer	This API performs read/write operation to the SPI Interface.

Description

This section lists all the USB-SPI bridging APIs.

4.5.1 MchpUsbSpiSetConfig Function

C

```
BOOL MchpUsbSpiSetConfig(
    HANDLE DevID,
    INT EnterExit
);
```

Description

This API enables/disables the SPI interface. If SPI control register is not edited by the user then this function would put SPI in default mode i.e, mode0 and dual_out_en = 0. Speed is dependant totally on the strap options.

A INT variable EnterExit is used to identify if it is pass thru enter or exit.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device

EnterExit	Pass thru Enter or exit option
	1 : Pass thru Enter;
	0 : Pass thru Exit;

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

    //Enter into SPI Pass thru
if (FALSE == MchpUsbSpiSetConfig(hDevice,1))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf("MchpUsbSpiSetConfig Failed- %04xn",(unsigned int)dwError);
    exit (1);
}
```

4.5.2 MchpUsbSpiFlashRead Function

C

```
BOOL MchpUsbSpiFlashRead(
    HANDLE DevID,
    UINT32 StartAddr,
    UINT8* InputData,
    UINT32 BytesToRead
);
```

Description

This API reads bytes of data mentioned in the BytesToRead parameter from the SPI Flash memory region of the device starting at address mentioned in the StartAddr parameter. Before reading from SPI Flash,it will check for correct device Handle and Proper buffer length.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
StartAddr	Start Address of the SPI Flash from where read operation starts.
InputData	Pointer to the Buffer which contains the data to be read.
BytesToRead	No of Bytes to be read.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");
BYTE byReadFirmwareData[64 * 1024];
```

```
if(FALSE == MchpUsbSpiFlashRead(hDevice,0x0000, &byReadFirmwareData[0],0x0064))
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("nError: Read Failed %04xn",dwError);
    exit (1);
}
```

4.5.3 MchpUsbSpiFlashWrite Function

C

```
BOOL MchpUsbSpiFlashWrite(
    HANDLE DevID,
    UINT32 StartAddr,
    UINT8* OutputData,
    UINT32 BytesToWrite
);
```

Description

This API writes bytes of data as mentioned in the BytesToWrite parameter to the SPI Flash memory region from memory location as specified in StartAddr. Before Writing to SPI Flash,it will check for correct device Handle and Proper buffer length.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
StartAddr	Start Address of the SPI Flash from where write operation starts.
OutputData	Pointer to the Buffer which contains the data to be written.
BytesToWrite	No of Bytes to be written.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];
uint8_t  pbyBuffer[128 * 1024];

HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

ReadBinfile("spi_firmware.bin",pbyBuffer);
if(FALSE == MchpUsbSpiFlashWrite(hDevice,0x00, &pbyBuffer[0],0xfffe))
{
    printf ("nError: Write Failed:n");
    exit (1);
}
```

4.5.4 MchpUsbSpiTransfer Function

C

```
BOOL MchpUsbSpiTransfer(
    HANDLE DevID,
    INT Direction,
    UINT8* Buffer,
    UINT16 DataLength,
    UINT32 TotalLength
);
```

Description

This API is the low level SPI pass thru command read/write. All commands to the SPI interface are directed as SPI Pass thru write, SPI pass thru read is nothing but a XDATA read from a specified offset where the response is stored.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
Direction	This bit will indicate if it is a Pass thru read or write. Read = 1; Write = 0.
Buffer	Buffer containing the command/ data to be sent to the device in case of SPI pass thru write. In case of pass thru read this buffer is used to store the data recieved from the device.
DataLength	This field is the size of USB command OUT packet being sent to the firmware.
wTotalLength	The wTotalLength is utilized to mention the number of bytes the SPI flash will return for the pass thru command.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Example

```
CHAR sztext[2048];
uint8_t  pbyBuffer[128 * 1024];

HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

if (FALSE == MchpUsbSpiSetConfig(hDevice,1))
{
    printf ("MchpUsbSpiSetConfig failed");
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
UINT8 bySPIBuffer[4];
UINT8 byOpcodeGetJEDECID = 0x9f;
//Write 0x9f to get JEDEC ID, Datalen is 1
```

```
//Totally 4 bytes will be retrived as jedec id, give total length as 4
if(FALSE == MchpUsbSpiTransfer(hDevice,0,byOpcodeGetJEDECID,1,4))
{
    printf ("MchpUsbSpiTransfer failed");
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
//Read 4 bytes of JEDEC ID
 if(FALSE == libMchpUsbSpiTransfer(hDevice,1,bySPIBuffer[0],4,4))
{
    printf ("MchpUsbSpiTransfer failed");
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
if (FALSE == MchpUsbSpiSetConfig(hDevice,0))
{
    printf ("MchpUsbSpiSetConfig failed");
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
```

4.6 Flexconnect API

Functions

	Name	Description
	MchpUsbFlexConnect	This API is used to send the Flexconnect Cmd to device.

Description

Flexconnect refers to the feature in Microchip USB hubs, wherein the upstream port swaps its role with downstream port 1 and also vice versa at run time

4.6.1 MchpUsbFlexConnect Function

C

```
BOOL MchpUsbFlexConnect(
    HANDLE DevID,
    UINT16 Config
);
```

Description

This API is used to send the Flexconnect Cmd to device with config data as specified in Config.

This Config value is based on the Product, please refer Product Specification for more details.

Preconditions

MchpUsbOpenID should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
Config	Passed as wValue field of the Flexconnect SETUP Command.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```
    hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfully");

//To turn on Flexconnect with Port 2 & 4 disabled
//wValue 0x8454 DIS_P2 = 1 ; Disable Port 2
//DIS_P4= 1 : Disable Port 4
//FLEX_STATE = 1 : Enable Flexconnect
//HDD TMR = 100b : Timer 1 second
//Bit 15 = 1
```

```
if (FALSE == MchpUsbFlexConnect(hDevice,0x8454))
{
    printf ("MchpUsbFlexConnect failed");

    exit (1);
}
```

4.7 Programming APIs

Functions

	Name	Description
◆	MchpProgramFile	Program configuration file to the selected device ID

Description

This section lists all high level APIs which can be used for programming.

4.7.1 MchpProgramFile Function

C

```
BOOL MchpProgramFile(
    HANDLE DevID,
    PCHAR InputFileName
);
```

Description

This API will program the configuration file given as argument to the selected device ID.

Preconditions

MchpUsbOpenID should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device
InputFileName	Input configuration file to be programmed into the device

Example

```
CHAR sztext[2048];

uint8_t  pbyBuffer[128 * 1024];


HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;


hDevice = MchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = MchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    exit (1);
}
printf("Device Opened successfullyn");

if(FALSE == MchpProgramFile(hDevice ,"MYcONFIG.BIN"))
{
printf("Programming Failed n");
dwError = MchpUsbGetLastErr(hDevice);
printf ("Error,%04xn",dwError);
exit (1);
}
```

4.8 Miscellaneous APIs

Functions

	Name	Description
	MchpUsbGetLastErr	Get last error for the specific hub instance.
	MchpUsbGetVersion	Get version no of the DLL.

Description

This section lists all miscellaneous APIs which contains various additional features.

4.8.1 MchpUsbGetLastErr Function

C

```
UINT32 MchpUsbGetLastErr(
    HANDLE DevID
);
```

Description

This API will get last error occurred when handling other API's in this library.

Preconditions

None.

Parameters

Parameters	Description
DevID	Handle to the device - Return value of MchpUsbOpenID.

Returns

Linux Error codes.

Remarks

None

Example

```
dwError = MchpUsbGetLastErr(hDevice);

//Print error here
cout << dwError << endl;
```

4.8.2 MchpUsbGetVersion Function

C

```
BOOL MchpUsbGetVersion(
    PCHAR pchVersionNo
);
```

Description

This API will get the version no of the DLL

Preconditions

None.

Parameters

Parameters	Description
pchVersionNo	Pointer to the buffer where the version number of the DLL will be stored.

Returns

None.

Remarks

None

Example

```
CHAR sztext[2048];
if (FALSE == MchpUsbGetVersion(sztext))
{
    printf ("nPress any key to exit....");
    exit (1);
}
//Print version number here
cout << sztext << endl;
```