Spectrometers · Spectrophotometers · Color Instruments · Spectrographs · Monochromators

111 Highland Drive · Putnam, CT · 06260 · USA
PHONE (860) 928-5834· FAX (860) 928-2676
http://www.spectralproducts.com

SDK for Linux

Manual

Spectrometers · Spectrophotometers · Color Instruments · Spectrographs · Monochromators

111 Highland Drive · Putnam, CT · 06260 · USA
PHONE (860) 928-5834· FAX (860) 928-2676
http://www.spectralproducts.com

Table of Contents

1. Development Environment

Raspberry Pi3
========================
OS: Raspbian Debian8-jessie
Kernel: Raspberrypi 4.4.11-v7+
Compiler: gcc – Rasobuab 4.9.2-10
========================

*** Required installation libraries**

- You need to install "libusb-1.0.x" to use Spectrometer driven API functions.
Install with the following command after connecting to the Internet:
Command: sudo apt-get install libusb-1.0

2. File Organization and Build

1) Lib

Folder containing object file of API functions for running Spectrometer - SPdbUSBLinux.o

AR archive file for file creation:
Command: ar crv libSPdbUSBLinux.a SPdbUSBLinux.o

2) Driver

Use "udev" to automatically create the node and mount the device.
The folder contains the 88-spusb-smSeries.rules file and the .hex, .config files by model. The above file should be copied to the specified path to normally recognize the device.

*** How to install**

- Auto Install: Run the executable file named **"smxxxinstall"** in the example code folder.
Installation (xxx is the Model information)
- Manual Install: Copy each file to the path specified by the following command:

```
cp 88-spusb-smSerise.rules /etc/udev/rules.d
cp SMXXX.hex /lib/firmware
cp SMXXX.conf /etc
```

.rules file: Setup rules when spectrometer is connected
.config file: device VID, PID information by model

.hex file: Driver file for communication with spectrometer

3) SDKs

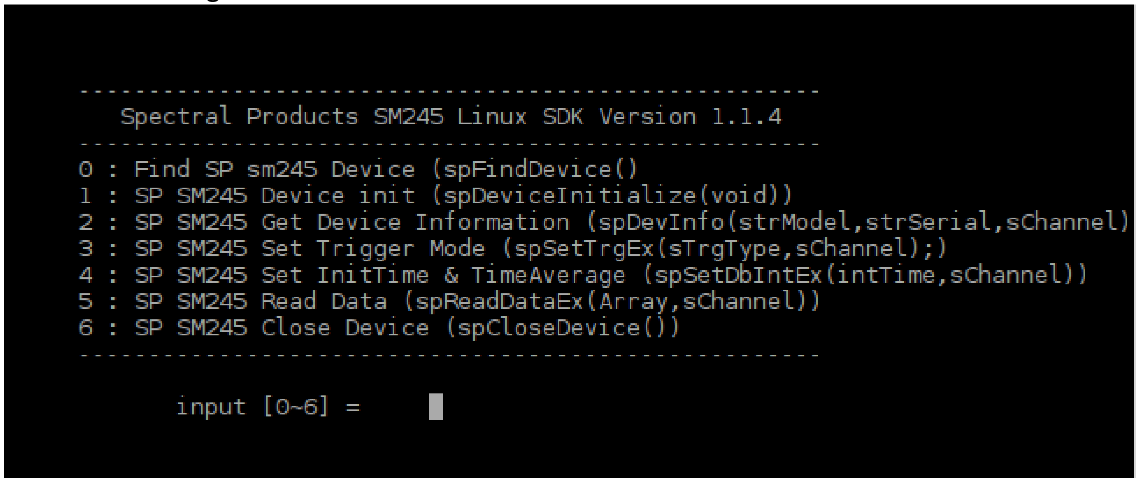
Sample code to load the SDK function and run spectrometer.
Attach as a compressed file by model (SM245, SM303, SM304, SM440, SM642)

*** How to Build**

- Make file: There is a makefile in the unpacked folder, move to that path in the terminal, and type "make"
- Manual Build: In the Terminal, type the following command:

```
ar crv libSPdbUSBLinux.a SPdbUSBLinux.o
gcc –o sdksmxxx sdksmxxx.c –L. –lSPdbUSBLinux –lm –lusb-1.0
```

When you build, "**AR archive file**" and "executable (**sdksmXXX**)" are created
Execute generated executable file as root.



```
----------------------------------------------------
   Spectral Products SM245 Linux SDK Version 1.1.4
----------------------------------------------------
0 : Find SP sm245 Device (spFindDevice()
1 : SP SM245 Device init (spDeviceInitialize(void))
2 : SP SM245 Get Device Information (spDevInfo(strModel,strSerial,sChannel)
3 : SP SM245 Set Trigger Mode (spSetTrgEx(sTrgType,sChannel);)
4 : SP SM245 Set InitTime & TimeAverage (spSetDbIntEx(intTime,sChannel))
5 : SP SM245 Read Data (spReadDataEx(Array,sChannel))
6 : SP SM245 Close Device (spCloseDevice())
----------------------------------------------------

       input [0~6] =
```

SM245 SDK Execution Screen

From 0 to 6, you can check the connection and function of spectrometer.

3. SDK Function

* Error Code

Error Code	Value	Description
SP_NO_ERROR	1	Normal Operation
SP_ERROR_DEVICE_IO_CTRL	-1	USB communication error
SP_ERROR_OPEN_DRIVER	-2	Handle open error
SP_ERROR_MEMORY_ALLOC	-5	Memory allocation error
SP_ERROR_NOTSUPORT_DEV	-7	Unsupported model
SP_ERROR_INPUT_PARAM	-10	Invalid input value
SP_ERROR_SHUTTER_VALUE	-13	Shutter operation input value error
SP_ERROR_WAIT_TIMEOUT	-258	Delay in response time
SP_EXT_TRG_WATING	-99	External Trigger input timeout

1) spFindDevice

```
short spFindDevice
(

)
```

This function is used to find the Spectrometer among the connected USB devices and test the connection.
Search Spectrometers in the order they are connected and allocate channels.

RETURN
If the function works normally, it returns the number of connected spectrometers, otherwise it returns a negative number.

2) spDeviceInitialize

```
short spDeviceInitialize
(

)
```

This function is used to set and verify the connection of all connected spectrometers. USB communication related setting function, so you can use the function of Spectrometer after using this function.

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

3) spSetIntEx

```
short spSetIntEx
(
        long lIntTime
        short sChannel
)
```

This function is used to set the light exposure time.

lIntTime: It is set in units of ms for Sony CCD and Hamamatsu PDA, and 10us for Toshiba CCD.
Sony CCD and Hamamatu InGaAs range from 1 to 65535
Hamamatsu Back-thinned ranges from 7 to 65535
Toshiba CCD range from 1 to 6553500
sChannel: Channel number of Spectrometer to operate

We recommend using the spSetDBIntEx() function when the input unit of Toshiba CCD is different, since it is difficult to use.

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value.

4) spSetDblIntEx

```
short spSetDblIntEx
(
        double dIntegration_time
        short sChannel
)
```

This function is used to set the light exposure time.

dIntegration_time: Input unit is integrated by ms.
Toshiba CCD sensor 0.01 ~ 65535.0
Sony CCD and Hamamatsu InGaAs range from 1.0 ~ 65535.0
Hamamatsu Back-thinned is from 7.0 ~ 65535.0
Toshiba rounds to two decimal places, and the other CCDs do not reflect decimal points.

sChannel: Channel number of Spectrometer to operate

<u>RETURN</u>
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

5) spReadDataEx

```
short spReadDataEx
(
        long *pArray
        short sChannel
)
```

This function reads CCD signal data through USB board

pArray: This is the memory address of the array that returns the read CCD data. The memory size must be greater than or equal to the number of pixels defined in the spDeviceInitialze () function.

SP_CCD_PIXEL_PDA = 1056
SP_CCD_PIXEL_G9212 = 512
SP_CCD_PIXEL_SONY = 2080
SP_CCD_PIXEL_S10420 = 2080
SP_CCD_PIXEL_TOSHIBA = 3680

Some detectors contain dummy pixels or optical blank pixels, so the actual number of pixels are:

SP_CCD_PIXEL_PDC_REAL = 1024
SP_CCD_PIXEL_G9212_REAL = 512
SP_CCD_PIXEL_SONY_REAL = 2048
SP_CCD_PIXEL_S10420_REAL = 2048
SP_CCD_PIXEL_TOSHIBA_REAL = 3648

In the case of Sony and Toshiba, 32 pixels from the first pixel are optical blank, and from the Hamamatsu Back-thinned, the first to 10 pixels are optical blank.
sChannel: Channel number of Spectrometer to operate

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

6) spPolyCalc

```
short spPolyCalc
(
        double *coefs
        short order
        double x
        double *y
)
```

This function calculates the following formula:

y=a0 + a1*x1 + a2*+x2 +...+aN*xN

coefs: Pointer to array of polynomial coefficients calculated using spPolyFit
order: The degree of a polynomial. In most cases, use a third-order polynomial calculation that is optimal for wavelength correction**.**
x: Calculated Pixel number
y: Calculated value

RETURN
None

7) spPolyFit

```
short spPolyFit
(
        double *x   // Array of independent variables
        double *y   // Array of dependent variables
        short numPts         // Number of points in independent and dependent arrays
        double *coefs        // Pointer of array to hold calculated coefficients [index: 0 -
                                order]
        short order// Order of polynomial
}
```

This function is used to find a polynomial function that calculates the wavelength per pixel. This function is for calibration and is calculated using the calibration data. Calibration data can be obtained using a calibration light source or a light source that transmits any narrow band filters.
It is the data which matches the reacting pixel by measuring the light which knows the wavelength with the spectrometer.
Calibration data is entered in x and y, and the array index range is 0 to numPts-1.
The resulting polynomial coefficients are output to the coefs array and then used with spPolyCalc to calculate the wavelength of the pixel.

numPts: Number of points in calibration data
coefs: Array of polynomial coefficients. Range of 0 to order-1
order: The number of dimensions in the polynomial. In most cases, use a third-order polynomial calculation that is optimal for wavelength correction.

<u>**RETURN**</u>
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

8) spGetWLTable

```
short spGetWLTable
(
        double *dWLTable
        short sChannel
}
```

This function gets the wavelength table of the connected device. Used to simply import a wavelength table. Calibration data stored in EEPROM is calibrated by polynomial calculation inside.

dWLTable: Calculating wavelength value output by each pixel of an array
sChannel: Operating channel number of Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

9) spDevInfo

```
short spDevInfo
(
        char *strModel
        char *strSerial
        short sChannel
}
```

This function gets the model name and serial number of the connected device.

strModel: Model name of the output array
strSerial: serial number of the output array
sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

10) spGetModel

```
short spGetModel
(
        short sChannel
}
```

This function returns the model type of the connected device.

sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value
0: SM2XX
1: SM440
2: SM303
3: SM304
4: SM642

11) spSetTec

```
short spSetTec
(
        long lTEC    // Sets TE Cooling On/Off
        short sChannel
}
```

This function sets TE Cooling On / Off.

lTEC: 0 = off, 1 = on
sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

12) spSelectCF

```
short spSelectCF
(
        long lCF // Sets Capacity value as 1pF or 10pF
        short sChannel
}
```

This function sets the capacity value of the InGaAs array detector.

lCF: 0 = 1pF, 1 = 10pF
1pF can increase the sensitivity of the signal, but it is less stable than 10pF.
sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value.

13) spInsShutter

```
short spInsShutter
(
        short sChannel
}
```

This function checks whether the connected device has a built-in shutter.

sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value
0: Built-in shutter X
1: Built-in shutter O

14) spSetShutterPos

```
short spSetShutterPos
(
        short sShutter
        short sChannel
}
```

This function sets the position of the shutter.

sShutter: 0 = Open, 1 = Close
sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value.

15) spGetShutterPos

```
short spGetShutterPos
(
        short sChannel
}
```

This function returns the current shutter position.

sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value
0: Open
1: Close

16) spSetTrgEx

```
short spSetTrgEx
(
        short sTrgType
        short sChannel
}
```

This function is used to set the triggering mode. Triggering modes are Internal Mode and External Mode. If you use external trigger signal, set it to External Mode and call spReadDataEx ()to collect data according to trigger signal. If the trigger signal input time is exceeded, SP_EXT_TRG_WAITING (-99) is returned from the spReadDataEx () function.

sTrgType: 11 = SP_TRIGGER_INTERNAL
12 = SP_TRIGGER_EXTERNAL
sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

17) spSetIntMode

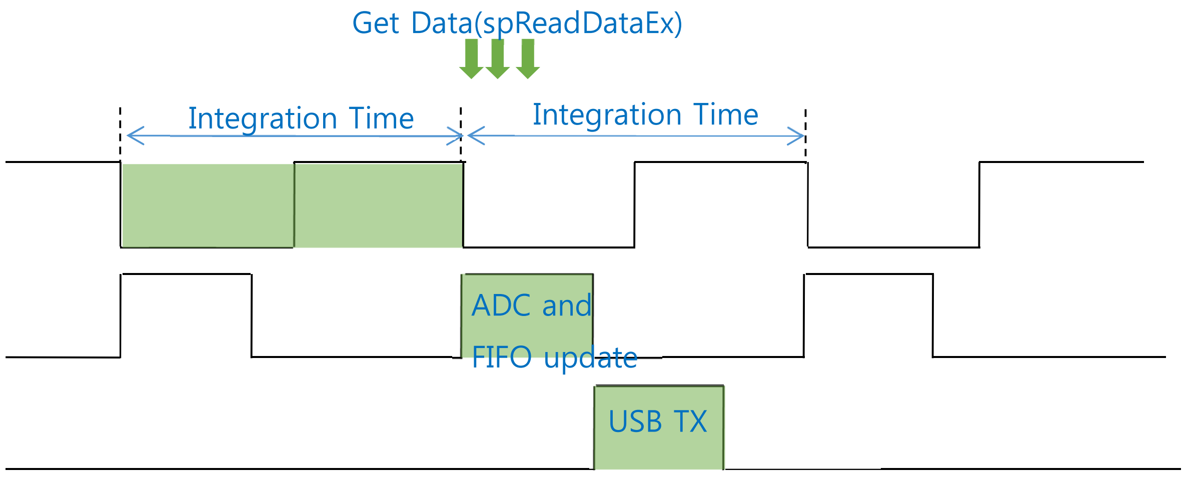
```
short spSetIntMode
(
        short ReadExMode
        short sChannel
}
```

This function sets the operation mode in the internal trigger mode and works only in the new version of SM245, SM303, and SM642 Spectrometer. There are three modes of operation: Software Trigger, Free run, Previous, and Free run Next. SM303 and SM642 support all operations and SM245 does not support free run Next mode. When SM245 is set to Free run Next mode, it is automatically set to Free run Previous mode from inside.

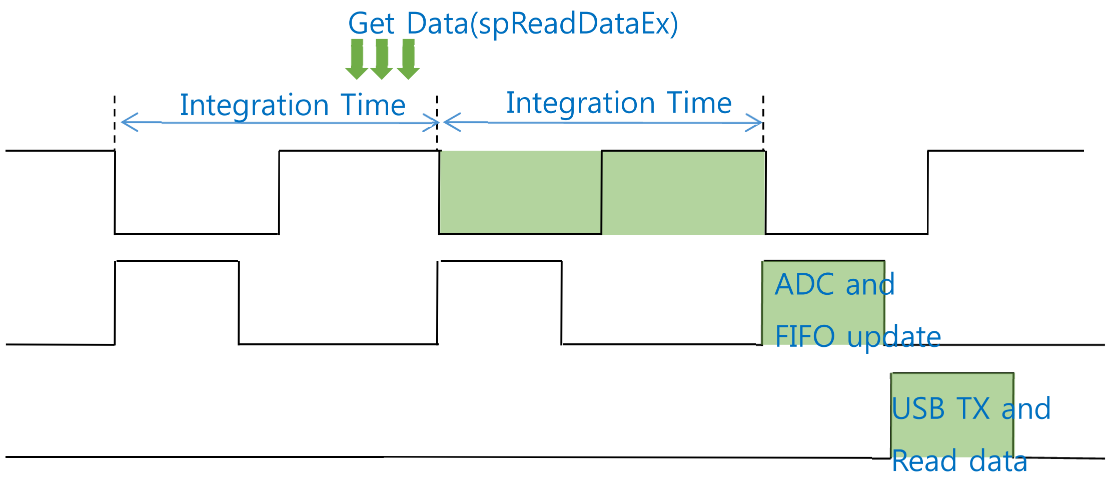
ReadExMode: 0 = SP_INTERNAL_SWTRIGGER
1 = SP_INTERNAL_FREERUN_PREV
2 = SP_INTERNAL_FREERUN_NEXT
sChannel: Channel number of operating Spectrometer

RETURN
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

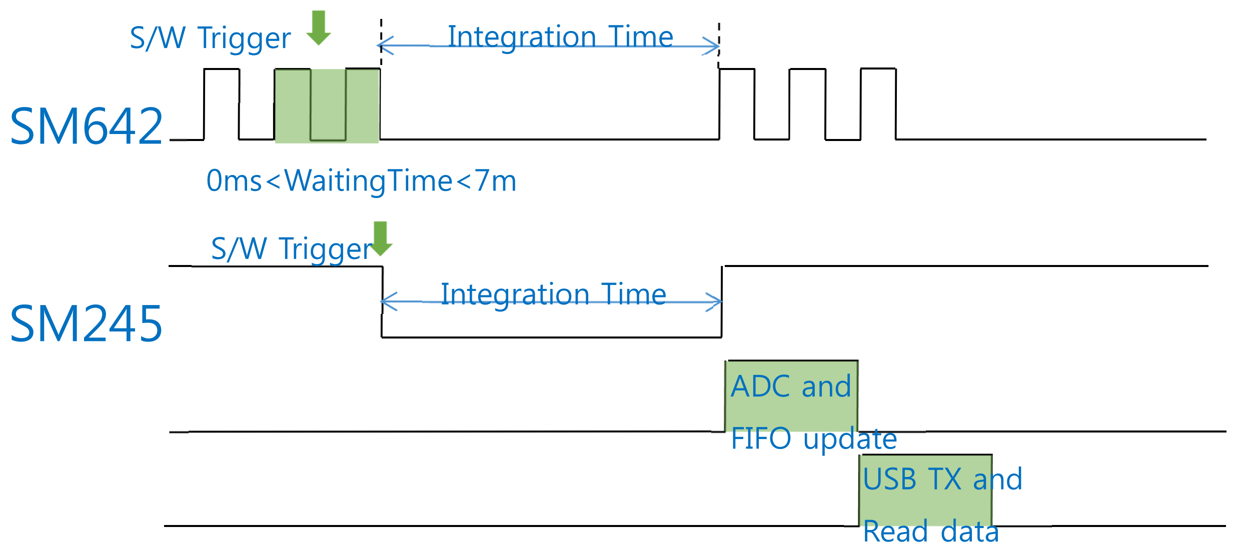
*** Timing Chart for each mode**
Free run Previous (SP_INTMODE_FREERUN_PREV = 1)



Free run Next (SP_INTMODE_FREERUN_NEXT = 2)



Software Trigger (SP_INTMODE_SWTRIGGER = 0)



SM245, SM642 Comparison Table

		SM245	SM642
Trigger Mode	Free run Prev	O	O
	Free run Next	X	O
	S/W Trigger	O	O
Minimum Integration Time		Free run Mode: 7ms S/W Trigger Mode: 1ms	All Modes: 7ms
ADC and FIFO update		5ms	6ms
USB Tx Time		Within 3ms	Within 3ms

18) spSetExtEdgeMode

```
short spSetExtEdgeMode
(
        short ExtrggerMode
        short sChannel
}
```

This function sets the interrupt method in Hardware Triggering Mode. Falling and rising are used in two ways, and default is Falling.

Ext: 0 = SP_INTERNAL_SWTRIGGER
1 = SP_INTERNAL_FREERUN_PREV
2 = SP_INTERNAL_FREERUN_NEXT

sChannel: Channel number of operating Spectrometer

<u>RETURN</u>
SP_NO_ERROR (1) if normal operation, otherwise returns negative value

19) spCloseDevice

```
short spCloseDevice
(
        short sChannel
}
```

This function is used to disconnect the USB board. Usually called when the application terminates.

sChannel: Channel number of operating Spectrometer

<u>RETURN</u>
SP_NO_ERROR (1) if normal operation, otherwise returns negative value.