NanEye USB3 API

Revision History:

Version	*Date*	*Modifications*	*Author*
1.0.0	16-01-15	Document creation	João Santos
1.1.0	22-02-16	Update for the NanoUSB3 v1.0 and v1.1 different constructors	João Santos
1.2.0	19-07-16	Updated for NanEye USB3 cppCLI	João Santos

Table of Contents

Index of Tables

Index of Figures

1 NanEye USB3 API

In this API there is the NanEyeUSB3Demo project that is used to get data from the NanoUSB3 board.

1.1 NanEyeUSB3Demo

This project is the NanEye USB3 C# demo, that shows how to use the board/sensor in order to get the images from the sensor.

In this project, there is a need to make a reference to **AviFile.dll**, **Awaiba.dll**, **AwFrameProcessing.dll, GeneralFpgaAlgorithms.dll and NanEyeUSB3Provider.dll** as in Figure 1:

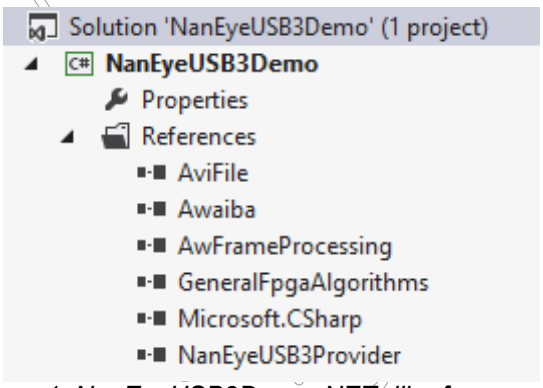

Figure 1: NanEyeUSB3Demo .NET dll references

Note: The files that are located in the **dependencies** folder need to be placed in the same folder as the application.

1.2 NanEyeUSB3 CppCLI

This project is similar to the C# project, but using the Cpp/CLI language.

It needs to have a reference to Awaiba, AwFrameProcessing, GeneralFpgaAlgorithms and NanEyeUSB3Provider, as in figure 2:

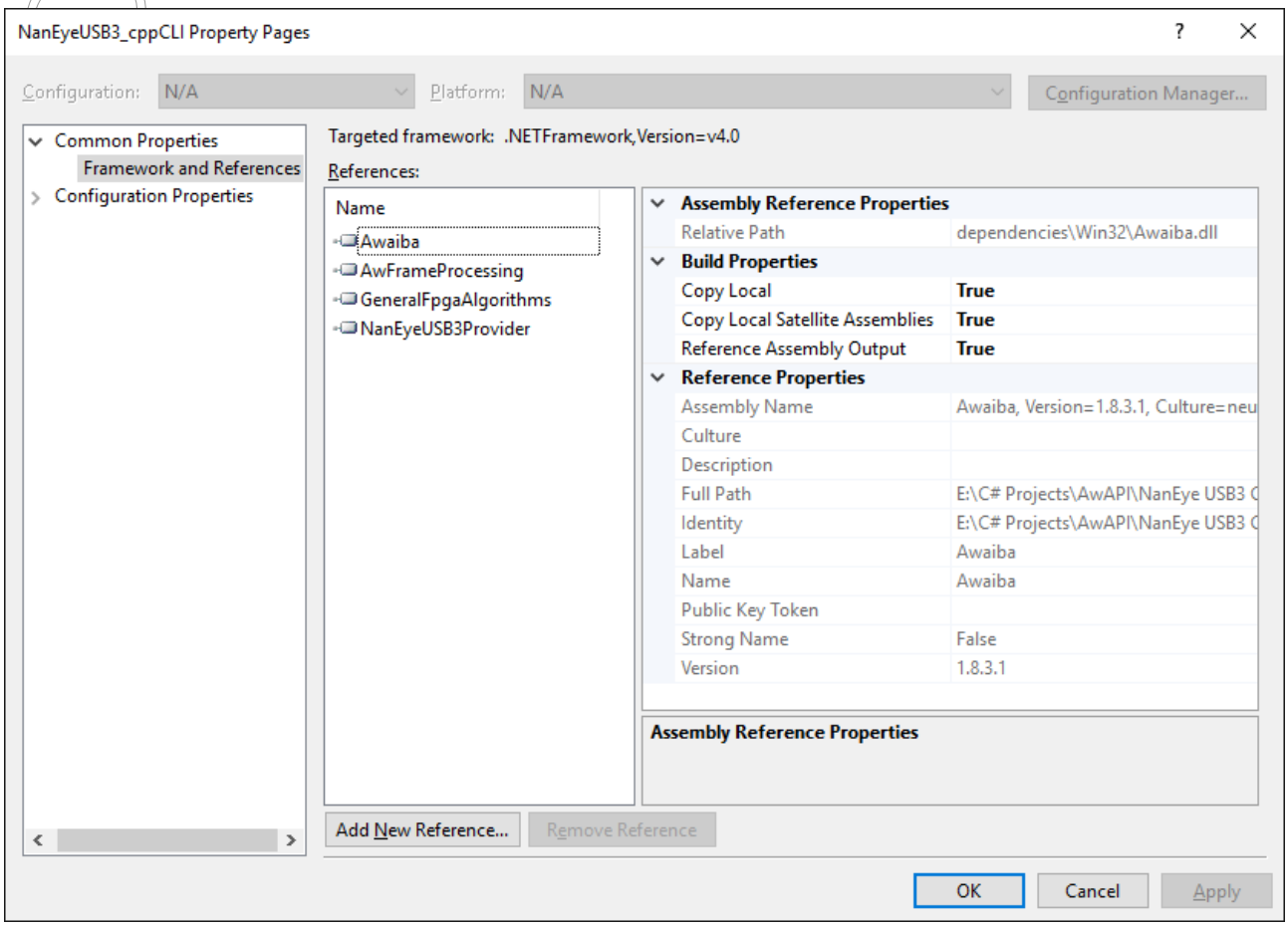

Figure 2: References in CppCLI project

2 NanEyeUSB3Demo - How to get images

After having everything configured from chapter 1.1 , you can go to the Form1.cs file, and check its code.
In this file you have a *macro* definition of either the “NanoUSB3_v1_0” or “NanoUSB3_v1_1”, and with that macro you can use the code for either of the adapter boards.

2.1 Initialization

2.1.1 Constructor

To start, you should create a NanEyeUSB3Provider instance. At the moment we have two different adapter boards, so you should check which one you have in order to use the correct constructor:

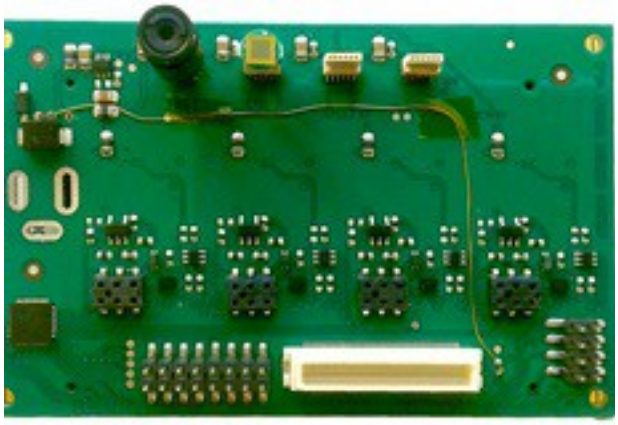

Figure 3: NanoUSB3 v1.0

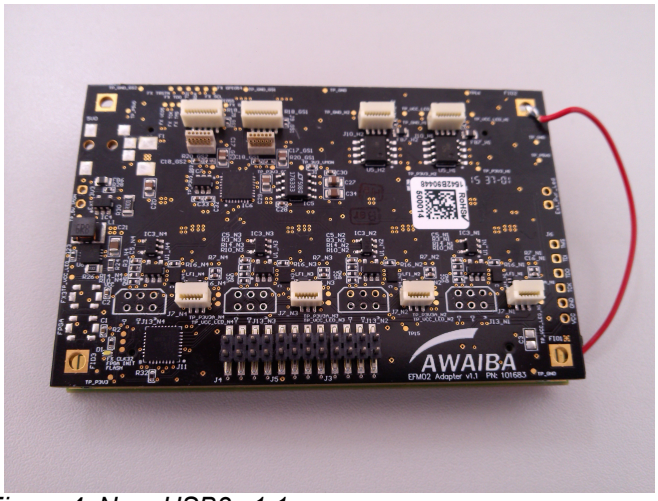

Figure 4: NanoUSB3 v1.1

If you have the **NanoUSB3 Adapter 1.0** (figure 3) , you should use this constructor:

```
NanEye2DUSB3Provider provider = new NanEye2DUSB3Provider();
```

If you have the **NanoUSB3 Adapter 1.1** (Figure 4) then you should use this constructor:

```
NanEye2DUSB3_2Provider provider = new NanEye2DUSB3_2Provider();
```

Note: In the **CppCLI** language, when creating the cosntructor, add the "^" when instantiating the class.

2.1.2 Configuration Files

After creating the instance, you should configure the "FPGA" and "FW" file to use. For this, you should use the **SetFpgaFile** and the **SetFWFile** methods:

```
provider.SetFpgaFile(@"Fpga Files\NanEye_efm02 - XC6SLX45 v1.1.1.bin");
provider.SetFWFile(@"Fpga Files\fx3_fw_30EP.img");
```

2.1.3 Sensor Configuration

Then, you need to say from which sensors you want to get data from (up to 4 sensors simultaneously). On the code, there is used a **List** of **bool** values with either **false** or **true** according to if it is to get data from that sensor:

```
List<bool> sensorsReceive = new List<bool>();
sensorsReceive.Add(true);  //J1
sensorsReceive.Add(false); //J2
sensorsReceive.Add(false); //J3
sensorsReceive.Add(false); //J4
```

On the **sensorsReceive** List, the user will receive images from the sensor at J1.

Then, you should put the **sensorsReceive** variable in the **Sensors** property, as stated below:

```
provider.Sensors = sensorsReceive;
```

2.1.4 Configure Automatic Exposure Control Hardware

After this, you need to create four instances of "Automatic Exposure Control Hardware", that will allow you to control the Automatic Exposure Control algorithm, and its properties.

Important: By default this algorithm is active, so you need to have this configured in order to change manually the sensor registers.

To create the list of instances:

```
List<AutomaticExposureControlHardware> aec = new List<AutomaticExposureControlHardware>();
```

Then, you should use a cycle in order to create all the required instances:

```
for(int i=0; i<4; i++)
{
    aec.Add(new AutomaticExposureControlHardware());
    aec[i].SensorId = i;
}
```

Important: Even if you have only one sensor plugged, you need to create all the four instances of the class.

Then, you assign this created instances to the provider you created firstly:

```
provider.SetAutomaticExpControl(aec);
```

2.1.5 Image Processed and Exception Events

The user can set the ImageProcessed event to get data. Now, the **provider_ImageProcessed** function is called when a new image arrives.

```
provider.ImageProcessed += provider_ImageProcessed;
```

The user can set the exceptions event to receive all the exceptions that are raised below:

```
provider.Exception += provider_Exception;
```

2.2 Automatic Exposure Control

After creating the "AutomaticExposureControlHardware "instances, in 2.1.4 , the user can define the AEC parameters as defined in the algorithm's documentation. The whole documentation regarding this can be found on our website, on the NanEye-AutomaticExposureControl document.

2.3 Sensor

To change the NanEye's registers, you should use the **WriteRegister** method.
This method receives a `NanEyeUSB3RegisterPayload` instance, that has the register value and address. The list of addresses and values are used as in table 1:

Setting	Address	Value Range	Default
Gain	0x01	[0-3]	2
Offset	0x02	[0-3]	3
Exposure	0x03	[250-0]	249
Vrst_Pixel	0x07	[0-3]	1
Vref_Cds	0x08	[0-3]	2
Digipot	0x04	[1700-2400]	1800
Led_State	0x05	True-False	False
Led_Value	0x06	[0-4096]	0

Table 1: Nan Eye Registers

2.4 Operations

The user can do some operations such as Start and Stop the sensor, check if the board is connected, check if the sensor is capturing. Those operations can be acessed as the properties of the provider instance. As for example:

To start capture images:

```
provider.StartCapture();
```

To stop capture images:

```
provider.StopCapture();
```

2.5 Getting Images

After calling the Start function, the sensor is now sending images and the user can now retrieve the images from it. For that it uses a `OnImageReceivedBitmapEventArgs` event that allows the user to grab the images from the sensor. It is set as in 2.1.5 and the function is used as below:

```
private void provider_ImageProcessed(object sender, OnImageReceivedBitmapEventArgs e)
{
	//Handle the image data
}
```

2.6 Image Processing

With the AwFrameProcessing reference, the user has access to the Awaiba's Frame Processing library. The whole documentation regarding this can be found on our website, on the **Awaiba Image Processing** document.