NM500 Software Development Guide

C/C++ Wrapper API for Windows

Version 1.0.8

Contents

1. Introduction

This document is intended to use with nepes NM500 Software Development Kit. The nepes NM500 Software Development Kit allows you to manage nepes NM500 neuromorphic chip and implement custom solutions by using libraries, utilities, and programming interfaces provided with the nepes NM500 Softwware Development Kit.

The nepes NM500 performs pattern recognition upon a knowledge model contructed with NeuroMem networks.

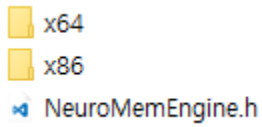
This document describes in detail how you can develop applications using the nepes NM500 Software Development Kit.

2. Set up Development Environment.

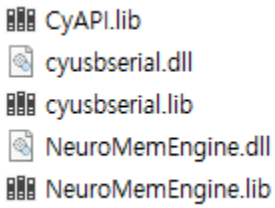
2.1 Overview

This nepes NM500 Software Development Kit contains NeuroMemEngine and USB libraries for NM500 application boards. A developer can use this SDK to create application that access NM500 neural network, build a knowledge model, and make decisions based on pattern recognition.

The SDK library contains the following files and directories that are required for building applications. It is fully supported on the 32-bit and 64-bit.



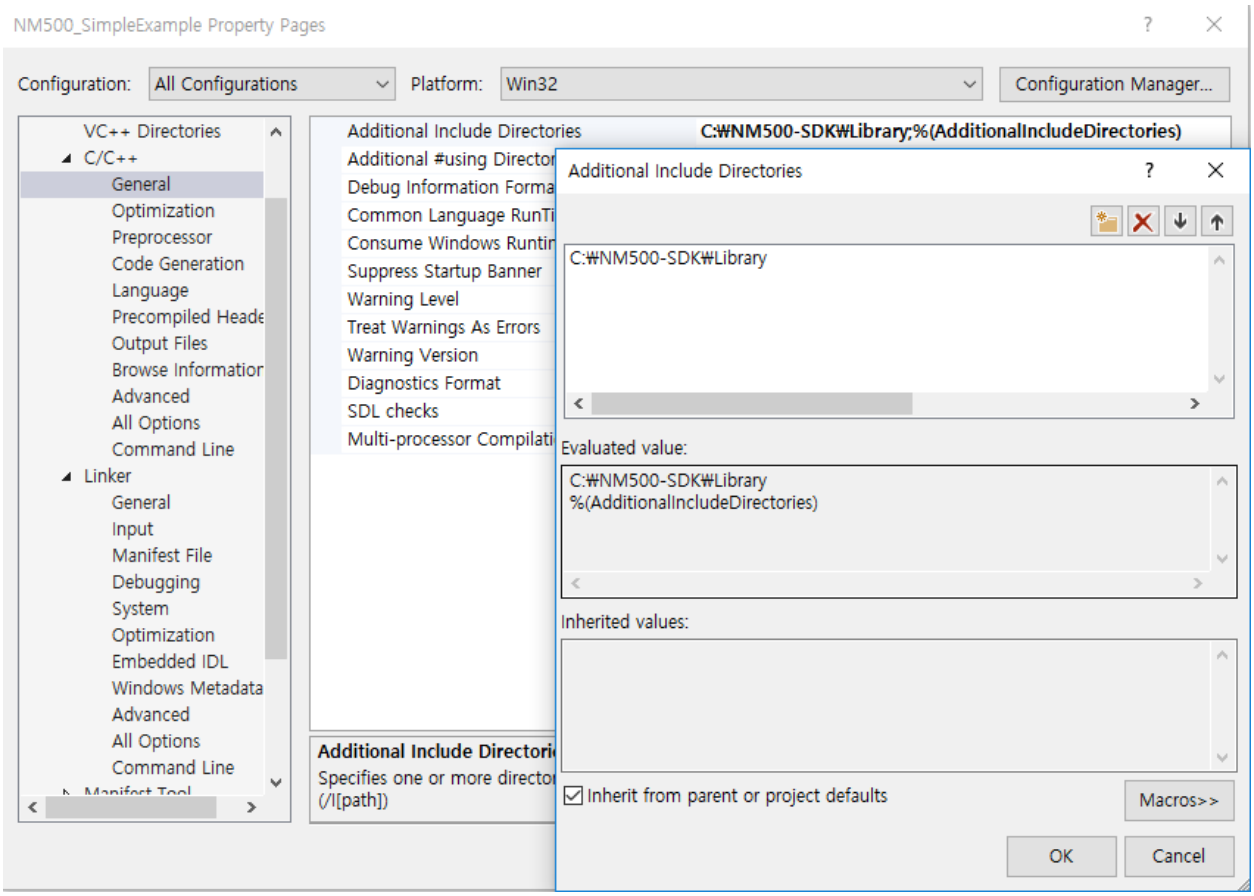
The included cy** USB libraries can only be loaded and accessed locally by NeuroMemEngine library



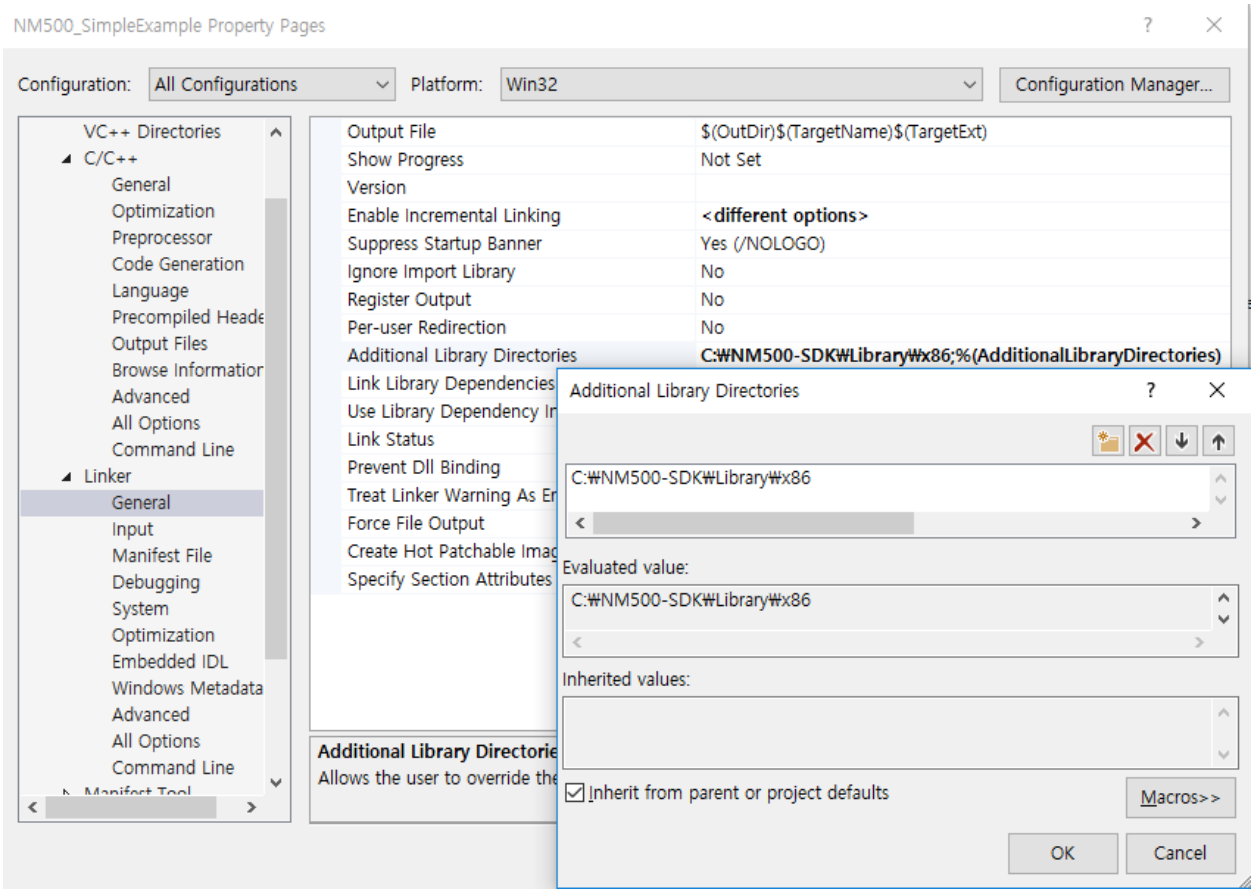
NeuroMemEngine library (DLL) is convenient high-level C++ wrappers.

2.2 Set up development environment in Visual Studio 2017

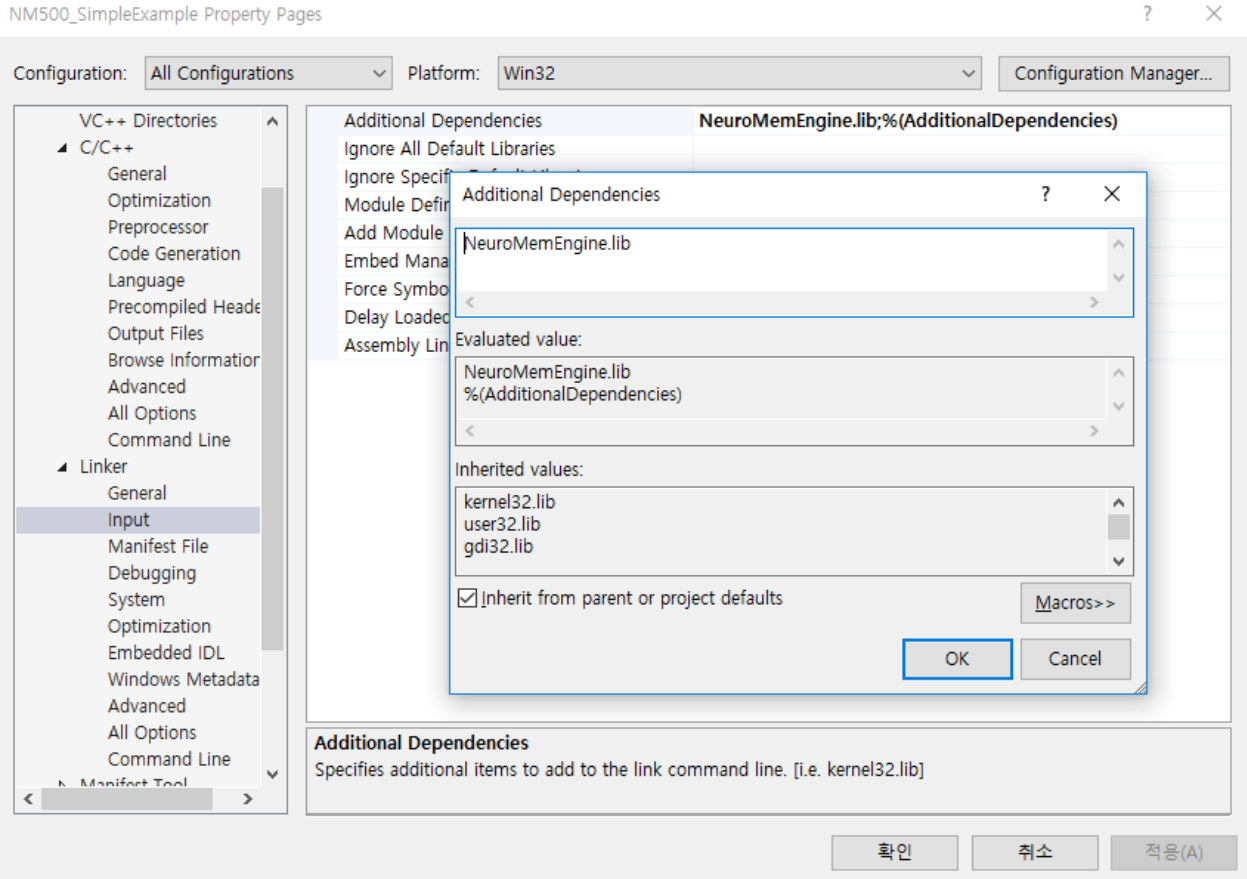
1) Set up the include directory. This is the directory that contains the header file "**NeuroMemEngine.h**", which describes the library interface.



2) Set up the library directory. This is the directory that contains the library files The library directory of SDK is divided into two directories: x86(32-bit) and x64(64-bit).



3) Enter library filename "**NeuroMemEngine.lib**" in additional dependencies for the libraries to use:



2.3 Getting started with Simple Example Project

The NM500 SDK includes a project that is configured to build a simple read/write demo.

3. NM500 APIs

3.1 GetDevices

The GetDevices function returns a list of the NM500 devices.

```
uint8_t GetDevices(NeuroMemDevice * devices, uint16_t count)
```

3.2 Connect

The Connect function is used to open selected device and performs the initialization of the NM500. It will return NM_ERROR_INIT_FAILED (100) if the NM500 is not properly initialized or selected device does not support the NM500.

```
uint16_t Connect(NeuroMemDevice * device)
```

3.3 GetNetworkInfo

The GetNetworkInfo function returns information about the NM500 neural network(NeuroMem) and device.

```
uint16_t GetNetworkInfo(NeuroMemDevice * device, NeuroMemNetworkInfo * info)
```

The information is as follows;

.

- **neuronMemorySize**: the memory size of the neuron - fixed at 256
- **neuronSize**: the total number of neurons on the NeuroMem network

- **version:** the version of the device firmware. Usually represented as X.Y

[Warning] This function performs the initialization of the NM500 to count the total number of neurons. Therefore, do not use this function during learning or recogition. This function is only useful if you want to check the overall network information at the beginning.

3.4 GetVersion

The GetVersion function returns the version of the device firmware. Usually represented as X.Y

```
uint16_t GetVersion(NeuroMemDevice * device)
```

3.5 GetNeuronCount

The GetNeuronCount function returns the number of neurons used on the NeuroMem network.

```
uint32_t GetNeuronCount(NeuroMemDevice * device)
```

The return value is the number of neurons actually used for learning, not the total number of neurons.

3.6 Forget

The Forget function is used to clear all knowledge on the NeuroMem network.

```
uint16_t Forget(NeuroMemDevice * device)
```

It will only initialize the daisy-chain(index of neuron) on the NeuroMem network. In other words, It will not erase the memory data(stored weight memory) of the neuron. Therefore, the neuron memory

data retrieved using readNeuron(s) function can contain the garbage data if you do not use maximum memory size of neuron as input vector size. (the maximum memory size of neuron is 256). However, it is faster than Reset function.

3.7 Reset

The Reset function is used to clear all knowledge on the NeuroMem network.

```
uint16_t Reset(NeuroMemDevice * device)
```

It performs the initialization of all the NeuroMem network setting and stored weight momory data of neurons.

3.8 SetNetworkType

The SetNetworkType function is used to set the NeuroMem network mode.

```
uint16_t SetNetworkType(NeuroMemDevice * device, uint16_t type)
```

The NeuroMem supports 2 types of network mode: RBF and KNN network. The RBF network must be selected to build a knowledge model.

Parameters

- type
  0: RBF network mode, 1: KNN network mode

3.9 GetNetworkType

The GetNetworkType function returns information about current network mode. To change network mode is possible at any time during recognition(classification)

```
uint16_t GetNetworkType(NeuroMemDevice * device)
```

To change network mode is possible at any time during recognition(classification)

Returns

- 0: RBF network mode, 1: KNN network mode

3.10 GetNetworkStatus

The GetNetworkStatus function returns information about the current status of the NeuroMem network.

```
uint16_t GetNetworkStatus(NeuroMemDevice * device, NeuroMemNetworkStatus * status)
```

The information is as follows;

- **networkType**: the type of network mode: (0: RBF network mode, 1: KNN network mode)
- **networkUsed**: the number of neurons currently used
- **context:** the context ID that is currently used. It is same as global context ID
- **norm**: the type of norm (L1 or Lsup) that is currently used.

3.11 SetContext

The SetContext function is used to configure the current(global) context attributes on the NeuroMem network.

```
uint16_t SetContext(NeuroMemDevice * device, NeuroMemContext * context)
```

The parameters are as follows;

- **context:** the context ID that is used to split the NeuroMem network into subnetworks. The neurons can be associated to different contexts and their use can be enabled or disabled by selecting a context value
- **norm**: the type of norm (L1 or Lsup).
- **minif(Minimum Influence Field)**: It represents minimum differences for dissimilarity judgement and it is used to control uncertain domain.
- **maxif(Maximum Influence Field)**: it represents maximum differences for similarity judgement and it is used to adjust conservatism. The values range from 1 to 65535.

3.12 GetContext

The GetContext function returns information about the current context on the NeuroMem network

```
uint16_t GetContext (NeuroMemDevice * device, NeuroMemContext * context)
```

The information is as follows;

- **context:** the context ID that is currently used. It is same as global context ID
- **norm**: the type of norm (L1 or Lsup) that is currently used.
- **minif(Minimum Influence Field)**: It represents minimum differences for dissimilarity judgement and it is used to control uncertain domain. The default value is 2.
- **maxif(Maximum Influence Field)**: it represents maximum differences for similarity judgement and it is used to adjust conservatism. The values range from 1 to 65535. The default value is 16384(0x4000)

3.13 Learn

The Learn function is used to teach a neuron(s) with given vector on the NeuroMem network.

```
uint16_t Learn(NeuroMemDevice * device, NeuroMemLearnReq * data)
```

When a neuron learns input vector, its context information is set to the value of the current(global) context. If neurons have been assigned with the same context, it says that they will be run with same conditions, such as MinIF, MaxIF and Norm method.

When a new neuron is assigned to learn input vector, the NM_NCOUNT register value of the NM500 is incremented by one. If recognizable by existing neurons, new neurons are not allocated. Therefore, the NM_NCOUNT register value of the NM500 will not be changed.

The request data for learning is as follow.

```
typedef struct _NeuroMemLearnReq
{
        uint8_t vector[256];
        uint16_t size;
        uint16_t category;
        NeuroMemNetworkResult ns;

} NeuroMemLearnReq;
```

The value of NeuroMemNetworkResult variable represents the result status of the learning, and it is as follows.

- NM_LEARN_ALREADY_KNOWN(0)
- NM_LEARN_SUCCESS(1)

3.14 Classify

The Classify function is used to identify to which of a set of categories input vector belongs.

```
uint16_t Classify(NeuroMemDevice * device, NeuroMemClassifyReq * data)
```

The request parameters is as follows.

```
typedef struct _NeuroMemClassifyReq
{
        uint8_t vector[256];
        uint16_t size;
        uint16_t k;
        NeuroMemNetworkResult status;
        uint16_t matchedCount;
        uint16_t degenerated[CLASSIFY_MAX_K];
        uint16_t distance[CLASSIFY_MAX_K];
        uint16_t category[CLASSIFY_MAX_K];
        uint32_t nid[CLASSIFY_MAX_K];

} NeuroMemClassifyReq;
```

The k parameter means that returns number of neurons matched. and the default value is 9. If network mode is RBF, it can be less than 9 (k).

The value of NeuroMemNetworkResult variable represents the result status of the classification(recognition), and it is as follows.

- NM_CLASSIFY_UNKNOWN (0): It cannot be classified into the learned model.
- NM_CLASSIFY_UNCERTAIN (4): It can be classified into the learned model, but it is judged to be one or more other categories and is unclear, and additional judgment is necessary.
- NM_CLASSIFY_IDENTIFIED (8): It can be clearly classified into the learned model.

3.15 Read

The Read function returns the value of the NM500 register specified.

```
uint16_t Read(NeuroMemDevice * device, uint16_t reg)
```

3.16 Write

The Write function is used to set the value of the NM500 register specified directly.

```
uint8_t Write(NeuroMemDevice * device, uint16_t reg, uint16_t data)
```

For the NM500 registers accessible, refer to 4. NeuroMem Register.

3.17 ReadNeuron

The ReadNeuron function returns information about a neuron of specific ID.

```
uint16_t ReadNeuron(NeuroMemDevice * device, NeuroMemNeuron * neuron)
```

The information of each neuron is as the following.

NID, NCR, CAT, AIF, MINIF, MODEL(vector: stored weight memory data of neuron)

```
typedef struct _NeuroMemNeuron
{
        uint8_t model[256];
        uint16_t size;
        uint16_t ncr;
        uint16_t aif;
        uint16_t minif;
        uint16_t cat;
        uint32_t nid;

} NeuroMemNeuron;
```

If the length of the learned data is less than 256, you can specify the length of data to read through the size variable.

```
uint32_t nid = 2;
uint16_t vectorSize = 20;
NeuroMemNeuron neuron;
neuron.nid = nid;
neuron.size = vectorSize;
uint32_t neuronCount = 0;
neuronCount = NeuroMemEngine::ReadNeuron(device, &neuron);
printf("[Neuron] NID: %d₩t NCR: %d₩t CAT: %d₩t AIF: %d₩t MINIF: %d₩n", neuron.nid, neuron.ncr,
neuron.cat, neuron.aif, neuron.minif);
for (int j = 0; j < neuron.size; j++) {
        printf("%d ", neuron.model[j]);
}
printf("₩n");
```

3.18 ReadNeurons

The ReadNeurons function returns information about the knowledge model that is a list of all the neurons learned.

```
uint32_t ReadNeurons(NeuroMemDevice * device, NeuroMemNeuron * neurons, uint32_t count)
```

The following is an example of querying up to 10 neurons learned.

```
uint16_t vectorSize = 20;
NeuroMemNeuron neurons[10];
neurons[0].size = vectorSize;
uint32_t neuronCount = 10;
neuronCount = NeuroMemEngine::ReadNeurons(device, neurons, neuronCount);
for (int i = 0; i < neuronCount; i++) {
        printf("[Neuron] NID: %d₩t NCR: %d₩t CAT: %d₩t AIF: %d₩t MINIF: %d₩n", neurons[i].nid,
neurons[i].ncr, neurons[i].cat, neurons[i].aif, neurons[i].minif);
```

```
        for (int j = 0; j < neurons[i].size; j++) {
                printf("%d ", neurons[i].model[j]);
        }
        printf("₩n");
}
```

3.19 WriteNeurons

The WriteNeurons function is used to upload the knowledge model to NM500.

```
uint32_t WriteNeurons(NeuroMemDevice * device, NeuroMemNeuron * neurons, uint32_t count)
```

The following is an example of uploading the knowledge model after adding a dummy neuron.

```
// Read the number of the neurons committed.
uint32_t neuronCount = NeuroMemEngine::GetNeuronCount(device);
cout << "neuron count in WriteNeuron: " << neuronCount << endl;
neuronCount++;
NeuroMemNeuron * neurons = new NeuroMemNeuron[neuronCount];
neurons[0].size = vecterSize;

// Read the Knowledge Model (Neurons)
NeuroMemEngine::ReadNeurons(device, neurons, neuronCount - 1);

// Add a dummy neuron at the end of the list.
cout << "neuron count in WriteNeuron: " << neuronCount << endl;
neurons[neuronCount - 1].nid = neuronCount;
neurons[neuronCount - 1].ncr = neurons[neuronCount - 2].ncr;
neurons[neuronCount - 1].cat = 3;
neurons[neuronCount - 1].aif = 10;
neurons[neuronCount - 1].minif = neurons[neuronCount - 2].minif;
neurons[neuronCount - 1].size = vecterSize;

memset(neurons[neuronCount - 1].model, 0x00, 256);
```

```
neurons[neuronCount - 1].model[0] = 3;
neurons[neuronCount - 1].model[vecterSize - 1] = 2;

printf(">> write neurons %d₩n", neuronCount);
neuronCount = NeuroMemEngine::WriteNeurons(device, neurons, neuronCount);

// Read neurons (3.18)
```

3.20 PowerSave

The PowerSave function is used to set the power mode of the NM500. It immediately changes to the power saving mode.

```
uint32_t __stdcall PowerSave(NeuroMemDevice * device)
```

[Notice] It is recommended that you call PowerSave function during idle periods after using low-level functions such as the Read () and Write () functions.

4. Flash Memory APIs

This APIs is supported on certain application boards with flash memory capabilities.

4.1 UpdateFirmware

The UpdateFirmware function is used to upgrade firmware on application board.

```
uint32_t __stdcall UpdateFirmware(NeuroMemDevice * device, const char * path);
```

```
NeuroMemDevice * target = &ds[0];
uint16_t result = NeuroMemEngine::UpdateFirmware(target, "new_version.bit");

if (result == 0) {
	printf("₩nCompleted ... Please Reconnect The Device again");
}
else {
	printf("₩nFailed ... ");
}
```

5. Camera APIs

This APIs is supported on certain application boards with camera capabilities.

5.1 GetFrame

The GetFrame function returns camera frame data. The Camera APIs is only supported on the Prodigy board and it supports only 1280 x 720 resolution and the format of the camera frame data is YUV422

```
uint32_t __stdcall GetFrame(NeuroMemDevice * device, DataCameraFrame * frame);
```

The return data for camera frame is as follow.

```
typedef struct _DataCameraFrame
{
	uint8_t * data;
	uint32_t offset;
	uint32_t size;
} DataCameraFrame;
```

The following is an example of converting YUV422 to BITmap (RGB888).

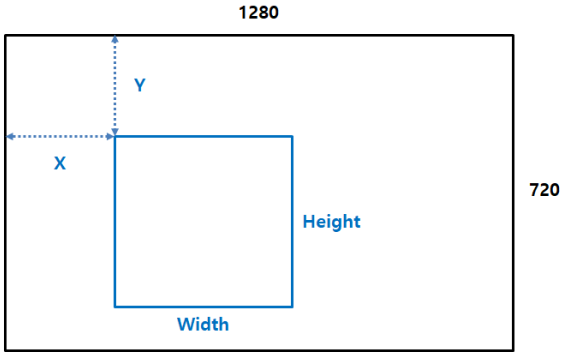
```
uint32_t size = width * height * 2;
uint32_t pages = size / 4;

int y0, y1, cb, cr;
int r, g, b;

int yuv422p = 0;
int index = 0;
int rgbIndex = 0;

for (int i = 0; i < pages; i++) {
```

```
        y0 = yuv[yuv422p++] & 0xFF;
        cb = yuv[yuv422p++] & 0xFF;
        y1 = yuv[yuv422p++] & 0xFF;
        cr = yuv[yuv422p++] & 0xFF;

        b = y0 + (1.370705 * (cr - 128));
        g = y0 - (0.698001 * (cb - 128)) - (0.337633 * (cr - 128));
        r = y0 + (1.732446 * (cb - 128));

        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;
        if (r < 0) r = 0;
        if (g < 0) g = 0;
        if (b < 0) b = 0;

        rgbValues[rgbIndex++] = r;
        rgbValues[rgbIndex++] = g;
        rgbValues[rgbIndex++] = b;

        b = y1 + (1.370705 * (cr - 128));
        g = y1 - (0.698001 * (cb - 128)) - (0.337633 * (cr - 128));
        r = y1 + (1.732446 * (cb - 128));

        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;
        if (r < 0) r = 0;
        if (g < 0) g = 0;
        if (b < 0) b = 0;

        rgbValues[rgbIndex++] = r;
        rgbValues[rgbIndex++] = g;
        rgbValues[rgbIndex++] = b;
}
```

5.2 SetROI

The SetROI function is used to set the ROI(Region of Interest) in camera resolution area.

```
uint32_t __stdcall SetROI(NeuroMemDevice * device, uint16_t x, uint16_t y, uint16_t width, uint16_t height);
```



6. NeuroMem Registers

Acronym	Description	Address	Normal mode	SR mode	16-bit default
NSR	Network Status Register	0x0D	RW	W	0x0000
GCR	Global Control Register	0x0B	RW		0x0001
MINIF	Minimum Influence Field	0x06	RW	RW	0x0002
MAXIF	Maximum Influence Field	0x07	RW		0x4000
NCR	Neuron Context Register	0x00		RW	0x0001
COMP	Component	0x01	W	RW	0x0000
LCOMP	Last Component	0x02	W		0x0000
INDEXCOMP	Component index	0x03	W	W	0x0000
DIST	Distance register	0x03	R	R	0xFFFF
CAT	Category register	0x04	RW	RW	0xFFFF
AIF	Active Influence Field	0x05		RW	0x4000
NID	Neuron Identifier	0x0A	R	R	0x0000
POWERSAVE	PowerSave	0x0E	W		n/a
FORGET	Forget	0x0F	W		n/a
NCOUNT	Count of committed neurons	0x0F	R	R	0x0000
RESETCHAIN	Points to the first neuron	0x0C		W	n/a
TESTCOMP	Test Component	0x08		W	0x0000
TESTCAT	Test Category	0x09		W	0x0000