NeuroShield User's Guide

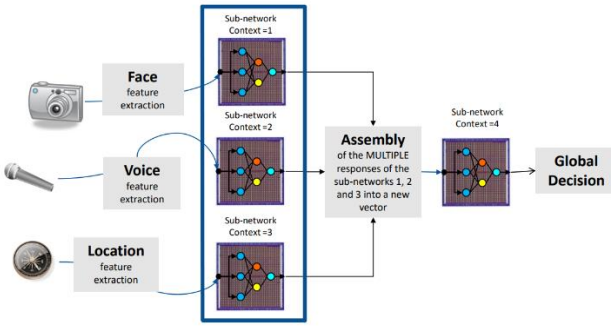
Version 1.0.5 , Feb 2018

Contents

1 Introduction

NeuroShield is a trainable pattern recognition board for IoT and smart appliances featuring the NeuroMem NM500 chip with 576 neurons which can interface with Arduino boards or a PC.

- ☐ The neurons learn by examples
  - ◘ Training can be done off-line or on the fly
- ☐ Continuous monitoring at low-power
- ☐ Can detect a novelty or anomaly
- ☐ Knowledge portability

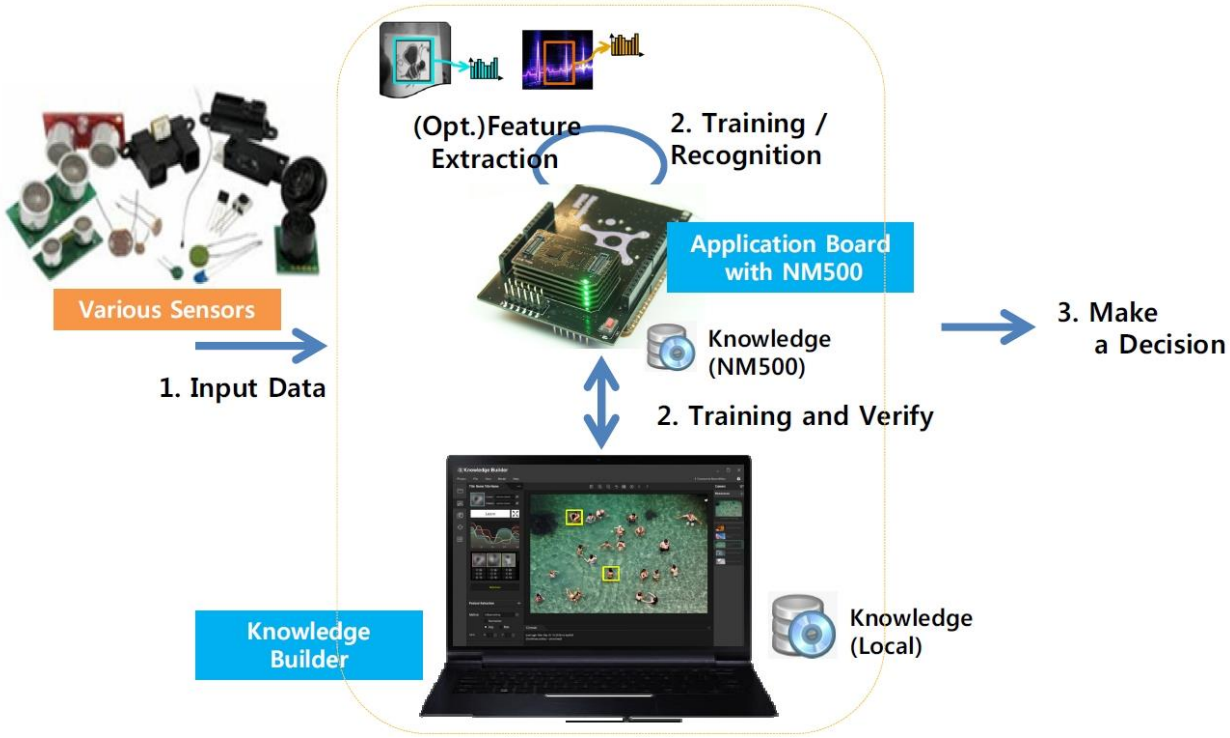


NeuroShield is easy to use through the NeuroShield library and Software Development Kit (SDK).

Data coming from a variety of sources can be converted into pattern vectors which are then broadcast to the neural network for either learning or recognition. All the user must do is focus on the quality of the input signals, and the selection of relevant and discriminant feature extractions.

NeuroShield manages the automatic learning of the examples and associated categories, the recognition of new patterns, detection of uncertainty cases (if any), the detection of drifts or decrease in confidence, and the reporting of novelties and/or anomalies.

Finally, the user must format the response of the neurons to convert it into a global decision or action for the application, without forgetting to save the knowledge built by the neurons for backup purposes or distribution to other systems.



2 Getting Started

Go to http://www.theneuromorphic.com and retrieve the Board Support Package which contains libraries and examples for various programming for the IDE.

Power supply

The NeuroShield is powered through the USB port or the Arduino power connector (J1). In term of IOs, NeuroShield only accepts 5V I/O on the Arduino connectors J2, J3 and J5.

※ If the IOREF on the main platform board is 3.3V, you should remove and use the NeuroShield IOREF pin under hardware version V0.3

SPI communication

Communication between NeuroShield and the Arduino/PC platforms is made through an SPI protocol described in this manual and the examples supplied with the board.
SPI communication supports up to 2MHz.

The default SPI pin assignment is the following:

- ☐ Arduino platform
  - ◘ SCK / Pin D13 (common)
  - ◘ MOSI / Pin D11 (common)
  - ◘ MISO / Pin D12 (common)
  - ◘ NM500_SSn / Pin D7 (for NM500)
  - ◘ SDCARD_SSn / Pin D6 (for SDCARD)
  - ◘ SPI_SEL* / Pin D5 (PC or Arduino selection)

* To activate an Arduino platform, SPI_SEL must be set to LOW
(Default pull up to HIGH on SPI_SEL pin)

I2C communication

I2C communication is used to control a motion sensor (MPU6050) on the NeuroShield. Both Arduino and PC control the sensor via this communication.

The default I2C pin assignment is the following:

- ☐ Arduino platform
  - ◘ SDA1 / Pin D14
  - ◘ SCL1 / Pin D15
  - ◘ SDA2* / Pin A4
  - ◘ SCL 2* / Pin A5

* To use SDA2 or SCL2 paths, refer to following a schematic as following

USB communication

The NeuroShield supports the USB port for a PC or Laptop, which is based on SPI protocol.
A Cypress chip converts USB to SPI interface, and the user can download an SDK from our website.

Testing

A series of short and academic test programs are supplied for the Arduino IDE. In addition to the short descriptions below, detailed comments are included in the source code and print statements executed by the code.

The NeuroShield has been tested with the following platforms:

Hardware Platform	Arduino/Mbed	PC or Laptop
IDE Platform	Arduino IDE/Mbed online compiler	Knowledge Studio/SDK*
Interface Boards	Kocoafab OrangeBoard series Arduino platform Board series Mbed platform series	

* Here is download link - http://www.theneuromorphic.com/products/knowledgestudio

Typical workflow

The functions of the NeuroShield library are described in Chapter 3.

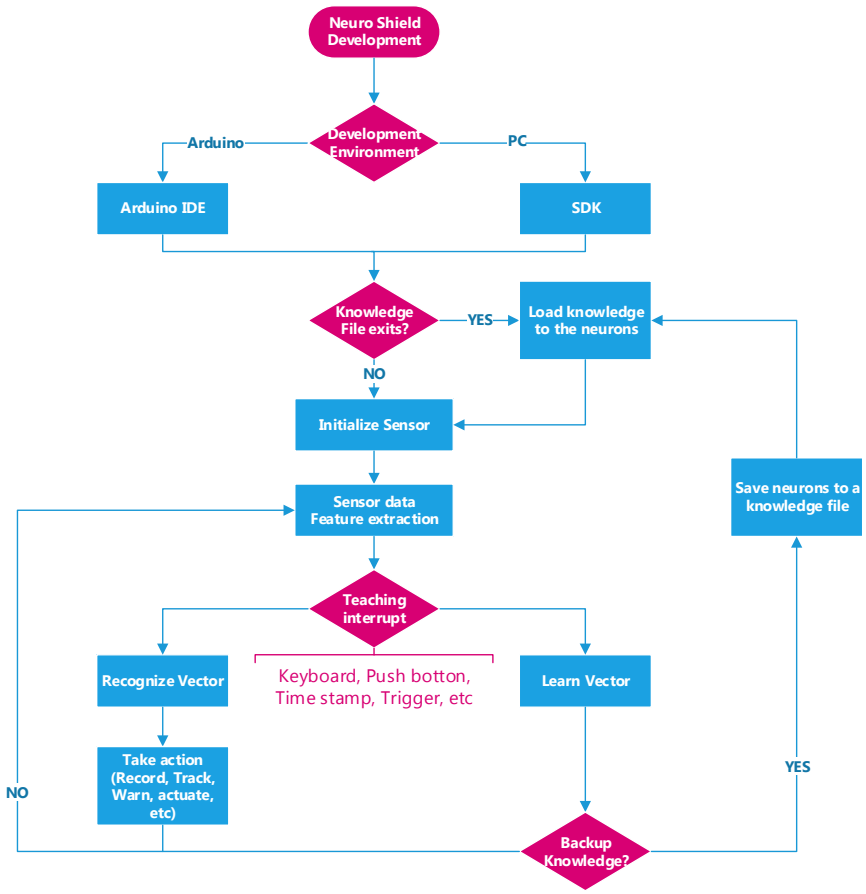
- ☐ Basic functions
  - ◘ Learn/Recognize vector
  - ◘ Save/Restore knowledge
- ☐ Additional functions
  - ◘ Tuning and expansion options

The data collected through the sensors can be broadcast to the neurons for learning and recognition.

Learning can be triggered by external user inputs, time stamps, but also the ability of the neurons to detect drifts from learned models. Learning can also be done "off line" on data previously collected and saved.

Recognition can consist of using the neurons to classify input patterns or identifying novel or abnormal patterns. Depending on the user application, the output of the neurons can control actuators, trigger a selective recording or transmission.

Possible applications of this technology include: identification, surveillance, tracking, and adaptive control.



How do the neurons learn and recognize?

The NeuroMem RBF tutorial demonstrates how the neurons build a decision space autonomously by learning from examples and recognize new vectors with the added ability to report cases of unknown and uncertain anomalies. For the sake of a simplicity the decision space is limited to a 2D space. But the mechanism is the same in recognizing an X1/X2 vector as it is to recognize all the way up to a X256 vector (which is the maximum length supported by the neurons in the NM500).

What is a feature vector?

The neurons are agnostic to the data source which means that they are ready to learn and recognize feature vectors extracted from text (parsing), discrete measurements, audio signal, video signal, and digital images. The only constraint is that the vector must be formatted as a byte array with a maximum length of 256.

In the case of images, a feature vector can be a subsampling of pixels within a region of interest, a histogram, or a histogram of gradients within a patch of pixels. In the case of an audio or biosensor signal, the feature vector can be a set of signal samples taken at a specific sampling rate, a histogram of peaks or zero crossing, or a power spectrum.

3 Function Libraries

Uint16_t begin();
Uint16_t begin(uint8_t slave_select)

This function clears the entire content of the neurons register and memory. The first neuron of the chain is then ready to Learn.

NeuroShield only:
This sets the SS (Slave Select) pin of the SPI BUS if the user wants to change the default value of D7.

void forget();
void forget(uint16_t maxif);

This function sets all the neurons to idle mode by clearing their category register. The first neuron of the chain is then ready to Learn.

Note: This function does NOT clear the memory of the neurons.

With this function the user can set the Maximum Influence Field (MAXIF) to a value different than the default setting of 0x4000.

The smaller the Maxif, the more conservative the generalization capability of the neurons will be whilst learning. The Maxif value range can be set between 2 and 0xFFFF.

uint16_t learn(uint8_t vector[], uint16_t length, uint16_t category);

Learning consists of broadcasting a vector and assigning its category-to-learn. Depending on the application, the category-to-learn can be defined by the user in a mode called Supervised Learning. It can also be assigned programmatically in a mode called Unsupervised Learning.

After the broadcast, the firing neurons with a category (other than category-to-learn), reduce their Influence Fields to exclude the input vector from their similarity domain. This is also when a new neuron is committed (if the non-firing neuron has the category equal to category-to-learn).

Inputs :
A vector is a sequence of bytes with a length between 1 and 256. This sequence can be derived from any type of data source such as; measurements, signal, sound, image or video. It can be composed of raw data or represent a signature extracted from raw data (also called a Feature).

A category value can range from between 1 to 32,767. The value of 0 is used to show the counter example (with the intention of correcting neurons firing erroneously, but without committing a new neuron).

Outputs :
This function returns the total number of committed neurons.

However, the user MUST do the following before calling this function:

- use the FORGET command to reset the network

- use the FORGET command and change the MAXIF value to learn in a more or less conservative manner

uint16_t classify(uint8_t vector[], uint16_t length);

uint16_t classify(uint8_t vector[], uint16_t length, uint16_t* distance, uint16_t* category, uint16_t* nid);
uint16_t classify(uint8_t vector[], uint16_t length, uint16_t K, uint16_t distance[], uint16_t category[], uint6_t nid[]);

The classification of a vector consists of broadcasting it to the neurons and reading the response of the firing neurons with successive readouts of; their distance, category, and if required, the neuron identifier registers.

This is when the "Winner Takes All" inter-connectivity between the neurons takes place so each neuron knows if it holds the next smallest distance and its algorithm is consequently the next closest match to the vector.

Depending on the user's application, they can decide to execute this readout only once and get the closest match, or to iterate and get a K neuron readout or the response of K's closest firing neurons.

It is also possible to limit the response to a simple classification status, which is known immediately after the end of a broadcast.

Inputs :
As mentioned with the previous function, a vector is a sequence of bytes with a length between 1 and 256. This sequence can be derived from any type of data source such as; measurements, signal, sound, image or video. It can be composed of raw data or represent a signature extracted from raw data (also called a Feature).

Outputs :
The first instantiations of this function returns the Network Status Register. Its lower byte can be decoded as follows:

Register	Description	Response Result
NSR[7:0]=0	the vector is not recognized by any neuron	UNKnown
NSR[7:0]=4	the vector is recognized but the firing neurons are in disagreement with its category	UNCertain
NSR[7:0]=8	the vector is recognized and all firing neurons are in agreement with its category	IDentified

The extended instantiation of this function returns a number of responses which is also the dimension of the distance, category and nid arrays. This number can be less than or equal to K if the network is set to RBF mode. This number is always equal to K if the network is set to KNN mode.

'Distance' is the distance between the input vector and the model stored in the firing neuron. It is calculated according to the selected norm (Bit 7 of the Global Context Register).

'Category' is the category of the firing neuron. It can range between 1 and 32,767. Its bit 15 is cleared, so the degenerated status of the neuron, if applicable, and is not reported.

In the extended instantiation of the function, the values of the distance array are in increasing order. The values of nid[k], distance[k] and category[k] are the neuron identifier of the k^{th} closest firing neuron, its distance register and category.

The user MUST do the following before calling this function:
- change the Network Status Register (NSR) to turn ON/OFF the KNN classifier
void readNeuron(uint16_t nid, uint8_t model[], uint16_t* context, uint16_t* aif,

uint16_t* category);
void readNeuron(uint16_t nid, uint8_t neuron[]);

The steps to saving the content of a specific neuron involves; switching the network to Save-and-Restore mode, pointing the specific neuron of the chain and reading sequentially its memory and registers.

Inputs:
NID is the identifier of the firing neuron. It can range from between 1 and the number of neurons available in the network. If greater than the number of neurons in the network, the content of the last committed neuron is returned.

Outputs:

Register	Description
Context	Neuron context register
Model[]	256 bytes of neuron memory*
AIF	Active Influence Field register
Category	Category register
Neuron[]	An array of 264 bytes

* Note: If an application is dealing with vectors of a length L lesser than 256, only the first L values of the Model[] are significant.

The order of the 264 bytes in an array are as follows:

Grouping	Description	Register
Bytes 0-1	Neuron Context Register	NCR
Bytes 2-257	256 bytes of neuron memory*	
Bytes 258-259	Active Influence Field	AIF
Bytes 260-261	Minimum Influence Field	MINIF
Bytes 262-263	Category	CAT

The remaining values might be irrelevant unless the Init function is executed prior to starting the Learning.

uint16_t readNeurons(uint8_t neurons[]);

The steps to saving the content of a specific neuron involves; switching the network to Save-and-Restore mode, pointing the specific neuron of the chain and reading sequentially its memory and registers.

Inputs
Neurons are an array of 8 bytes header and a dimension equal to NCOUNT records of 264 bytes:
- 8 + (NCOUNT * 256) bytes

NCOUNT is normally set to the same number as the committed neurons.

Outputs
This function returns the total number of committed neurons.

Each record must have the following format:

- Byte 0: 8, default header size
- Byte 1: 0, (discarded)
- Bytes 2-3: 256, neuron memory size
- Bytes 4-7: NCOUNT
- Bytes 8-271: 264 bytes array, repeat NCOUNT times

uint16_t writeNeurons(uint8_t neurons[]);

The steps to saving the content of a specific neuron involves; switching the network to Save-and-Restore mode, pointing the specific neuron of the chain and reading sequentially its memory and registers.

Execution of this function erases any previous knowledge held by the neurons.

Inputs
Neurons are an array of 8 bytes header and a dimension equal to NCOUNT records of 264 bytes:
- 8 + (NCOUNT * 256) bytes

NCOUNT is normally set to the same number as the committed neurons.

Each record must have the following format:

- Byte 0 : 8, default header size
- Byte 1 : 0, (discarded)
- Bytes 2-3 : 256, neuron memory size
- Bytes 4-7 : NCOUNT
- Bytes 8-271: 264 bytes array

Outputs:
This function returns the total number of committed neurons.

If this value is equal to 0xFFFF the network is full, which means that the neuron array exceeds the network capacity.

void setContext(uint8_t context);
void setContext(uint8_t context, uint16_t minif, uint16_t maxif);

This function allows the user to select the context and associated minimum and maximum influence fields (Minif and Maxif) for the next neurons to be committed.

The context can be associated to; a sensor, a type of feature extraction, a scale, or a combination of these parameters.

For more details refer to the NeuroMem Technology Reference Guide.

Inputs

Register	Description
Context	The context of the newly committed neurons
MINIF	Minimum influence field of the newly committed neurons
MAXIF	Maximum influence field of the newly committed neurons

void setRBF();

This function allows the user to set the neurons in Radial Basis Function mode (default mode).

void setKNN();

This function allows the user to set the neurons in K-nearest neighbor mode.

It also provides Read/Write access to the neuron registers:

NCOUNT, NSR, MINIF, MAXIF, GCR, DIST, CAT, NID, RSTCHAIN, AIF, and NID.

4 Example Code

A series of short and academic test programs have been developed for the Arduino Integrated Development Environment (IDE). In addition to the short descriptions below, detailed comments are included in the source code and print statements executed by the example code.

NeuroShield_SimpleScript.ino

This simple script stimulates the neurons to Learn and Recognize patterns generated programmatically.

The NeuroShield library includes (but is not limited to) the use of; the K-Nearest Neighbor (KNN) mode, the LSup/L1 norms for the calculation of distances, multiple contexts to Learn and classify vectors representing different dimensions (or data types) within the same KNN.

The user can practice with this functionality by switching the KNN from Radial Basis Function (RBF) mode to K-Nearest Neighbor (KNN) mode and comparing results between the two classifiers using a same knowledge and same input vector.

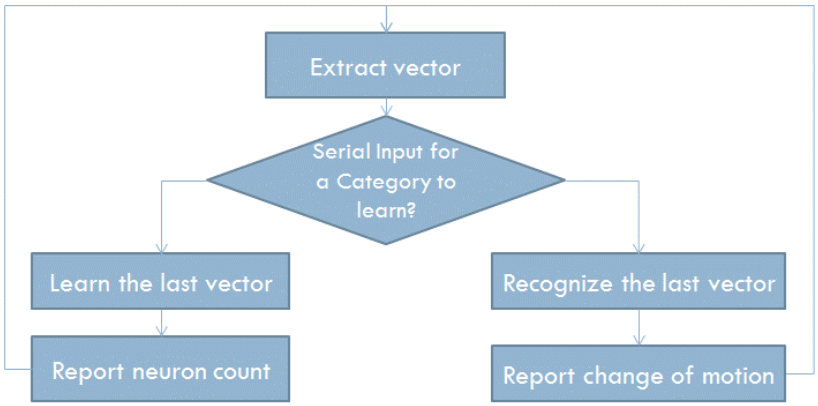
NeuroShield_andIMU.ino

This script requires the user to plug the NeuroShield into an Arduino board. It uses a 6-Axis accelerometer/ gyroscope Sensor (MPU6050) combined with the neurons of the NM500.

The script assembles signals from the accelerometer and gyroscope into a simple feature vector broadcasted continuously to the neurons for recognition. Learning is performed by entering a category value through the serial input.

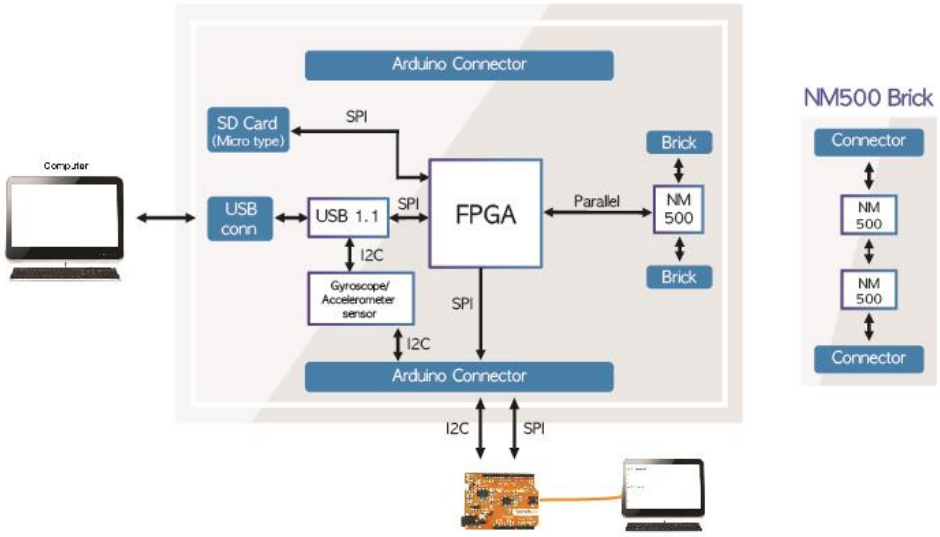
In the Arduino code, the LOOP continuously reads the IMU signals and extracts a simple feature vector which is a normalized subsampling of the 6-axis. The recognition of this vector starts automatically as soon as the neurons Learn some knowledge. The neurons build the knowledge as soon as the user starts giving this system some examples. And when the user enters a category through the serial port, the program associates it to the last feature vector.

The script mentions 3 categories (up/down, left/right and front/back) for the sake of simplicity, but the user can define their own categories combining not only a direction, but also other motion such as circular, or a range of amplitudes and speeds.

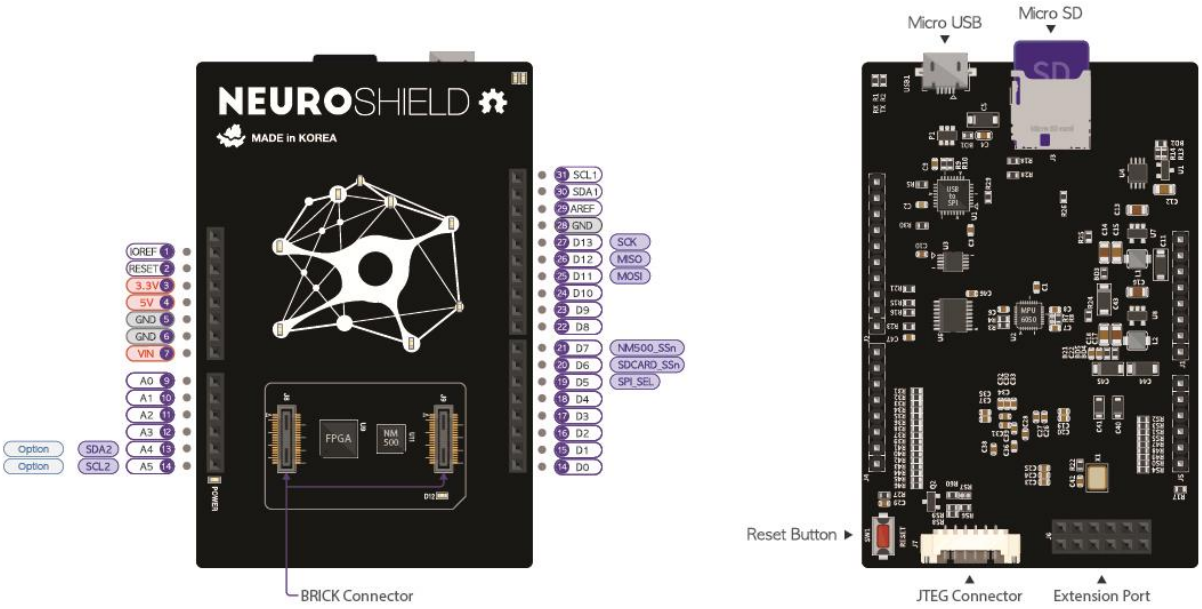


5 NeuroShield Description

Block Diagram



Hardware description



Arduino Power(J1)

Pins	Signal
1	NC
2	IOREF *
3	NC
4	3.3V
5	5V
6	GND
7	GND
8	Vin

Arduino digital (J2)

Pins	Signal (3.3V max)
1	D8
2	D9
3	D10
4	D11 /MOSI
5	D12 / MISO
6	D13 / SCK
7	GND
8	AREF
9	SDA
10	SCL

Arduino digital (J4)

Pins	Signal (3.3V max)
1	D0 (RX)
2	D1 (TX)
3	D2
4	D3
5	D4
6	D5 / SPI_SEL
7	D6 / SDCARD_SSn
8	D7 / NM500_SSn

Arduino analog (J5)

Pins	Signal
1	A0
2	A1
3	A2
4	A3
5	A4 / SDA(Opt.)
6	A5 / SCK(Opt.)

* If the IOREF on the main platform board is 3.3V, you should remove and use the NeuroShield IOREF pin under hardware version V0.3

NM500

NM500 is a NeuroMophic device that Learns and Recognizes user pattern data and achieves high data processing encompassed in its intrinsic parallel architecture giving it the following capability:

- Multi-modal (multiple sensor inputs sampled at the same time)
- Multi-scale/ Multi-feature (extracted from the same input signal to build redundancy)
- Reinforced learning - based on the recognition of past events

NeuroBrick (Neuron Extension Board) – Optional

If the user wants more neurons they can stack a NeuroBrick onto the NeuroShield. This device allows for a maximum of three NeuroBricks to be stacked at any one time.

The NeuroBrick itself has two NM500s, totaling 1152 neurons all together.

FPGA (Lattice-XO3)

This device makes all path bridging and coverts SPI to parallel protocol between the platforms and the NM500. A user can update the FPGA's own firmware via J7.

6-Axis accelerometer/gyroscope Sensor (MPU6050)

NeuroShield has a 6-axis Gyro/Accelerometer for the sensor application. As a test, users can use the basic example provided with this kit.

Arduino Connector

Communication between the NeuroShield and the Arduino board is through a Serial Peripheral Interface (SPI) BUS which is described below in Unit 6 of this manual.

Its SPI_Read and SPI_Write functions are included in the NeuroShield library.

USB

A PC or Laptop can control the NeuroShield with a USB but can also be developed using SDK (go to our website: http://www.theneuromorphic.com)

MicroSD Card

This allows the user's knowledge (Learned) data to be saved into an SD card.

LED

This allows user indication to show pre-defined LED scenarios.

6 SPI Protocol

Communication between the NeuroShield and the Arduino board is through a Serial Peripheral Interface (SPI) BUS protocol which is described below.

Its SPI_Read and SPI_Write functions are included in the NeuroShield library.

The SPI protocol is based on a 10-bytes control command described below and intended to access the various components of the board including the NeuroShield, NM500 chip and SD card.

Byte command sequence:

Grouping	Description
Byte 0 Byte 1-2-3-4 Byte 5-6-7 Byte 8-9 Byte +	Reserved R/W bit + 31-bit address 24-bit data length expressed in a number of words Minimum first 2 bytes of data Remaining data (up to data length)

Programming examples of the Read and Write functions are supplied in C++ and Arduino.

The programming uses the following convention in the examples supplied with this kit (but it can be re-written if the user requires):

- read(byte reg) if data length is a single word
- write(byte reg, int dat) if data length is a single word
- int write_vector(byte reg, byte* data, int size)

The full memory map of the NeuroSheild is located to the on-board NM500 chip or a chain of NM500 chips and will in time be published on our website: http://www.theneuromorphic.com

Address Range	**Module** = Address[30-24]	**Functionality defined by registers** = Address[23:0]	R/W	Default Value
0x01000000 0x0100001F	NM500(NeuroMem) Access	Access to the registers of the NM500*		
0x02000001	FPGA Access	FPGA Version check	R	0x01
0x02000002	FPGA Access	S/W Reset release to NM500	W	
0x03000000 0x03000007	FPGA Access	LED Scenario selection	W	

* Please refer to NM500 Users Manual

Write command

Reserved	Address[31:0]			Data length[23:0]	Data
0x00	Bit[31]=1	Module[6:0]	Register[24:0]	Length in words	Input array
1 byte	1 byte		3 bytes	3 bytes	Data length * 2 bytes

Single Write
Write 0x33AA to the MAXIF(register 7) in the NM500:

```
0x00 81 00 00 07 00 00 01 33 AA
```

Multiple Write
Write 4 consecutive byte components [11,12,13,14] for a neuron:

```
0x00 81 00  00 01 00 00 02 11 12 13 14
```

Read command

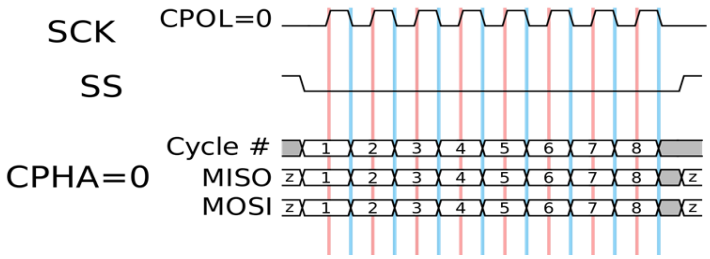
Reserved	Address[31:0]			Data length[23:0]	Data
0x00	Bit 31=0	Module[6:0]	Register[24:0]	Length in words	Output array
1 byte	1 byte		3 bytes	3 bytes	Data length *2 bytes

Single Read
Read the MINIF register 6 of the NM500 (module 1):

```
0x00 01 00 00  06 00 00 01 00 02
```

Data is returned into 2 bytes or a word

Timing Specifications



CPOL= 0 -> clock idle a logic level 0
CPHA= 0 -> Data sampled on rising edge of SCK

Recommended SCK frequency <= 2 MHz

7 Updating the FPGA firmware

For updating Lattice Field-Programmable Gate Array (FPGA) firmware, it requires:

(1) Lattice JTAG download cable
(2) Lattice FPGA programmer software

The software is downloadable from the Lattice FPGA website : www.latticesemi.com

Reprogramming through JTAG

The image below depicts the Lattice JTAG download cable set. It requires 4-pins for the JTAG standard (from Xilinx - a US based supplier of programmable logic devices).

J6 (FPGA Download Connector)	
No	Name
1	3.3V
2	TDO
3	TDI
4	TMS
5	TCK
6	GND

-

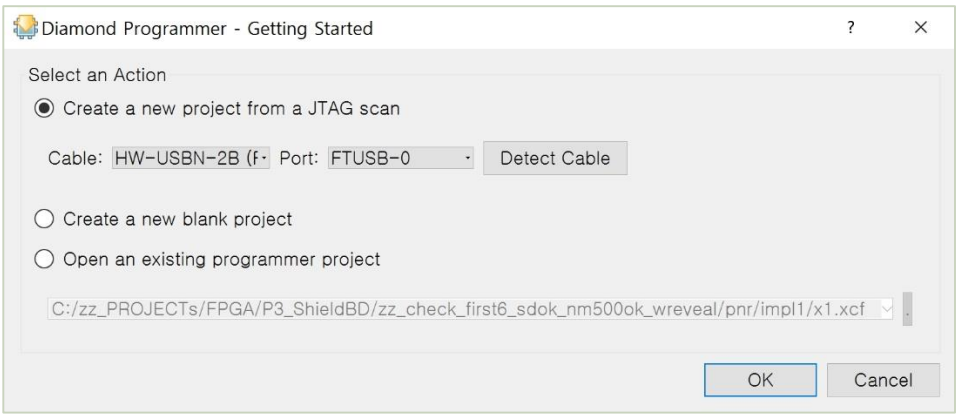
In the Lattice Design Tools suite of software the user has the choice to download the full product installation, 'Diamond' (free evaluation), or download, 'Diamond Programmer' (for free).

The Diamond Programmer is able to run as a standalone for immediate FPGA device configuration. The standalone product is intended for use in lab environments where the complete Lattice Diamond Tool (full product) is not required.

Directions are as follows:

- Open the Diamond programmer software
  (accessible through the Start Menu/Diamond Programmer)

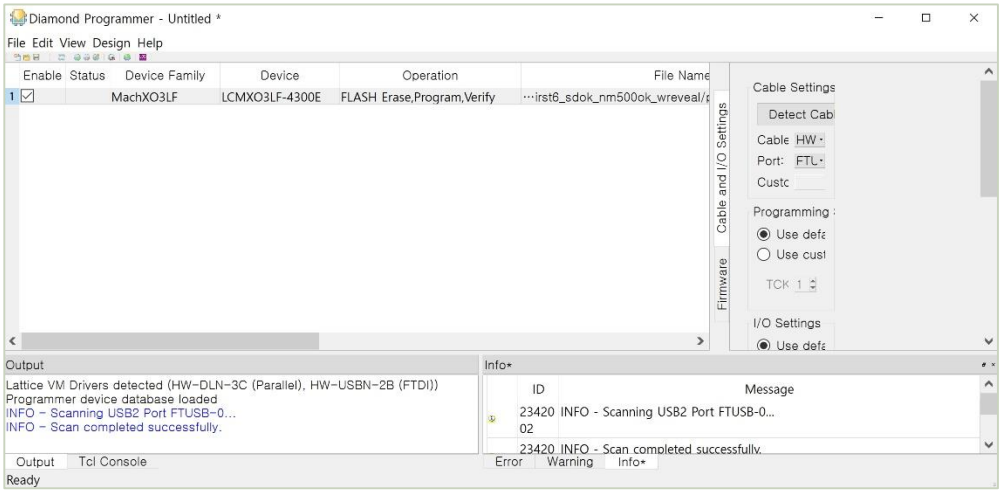
- Select the option "**Create a new project from a JTAG scan**" and click OK



- If by chance it fails to scan the first time, click **Detect Cable** button again to find a download cable:

The targeted FPGA chip has an internal flash memory to support its own firmware. As such it doesn't require an external flash memory device. Thus, it should be identified as the target device (LCMXO3LF-1300E) and displayed as below by:

- Selecting the firmware file **user.jed** in the File Name field (in the User folder)



- The user can now re-program it

- Clicking on the green icon **on top**

- Select **Program** from the **Design** top-down menu

- If it has carried out the procedure correctly the user can see a window-popup that reads:

- "**INFO - Operation: successful**"

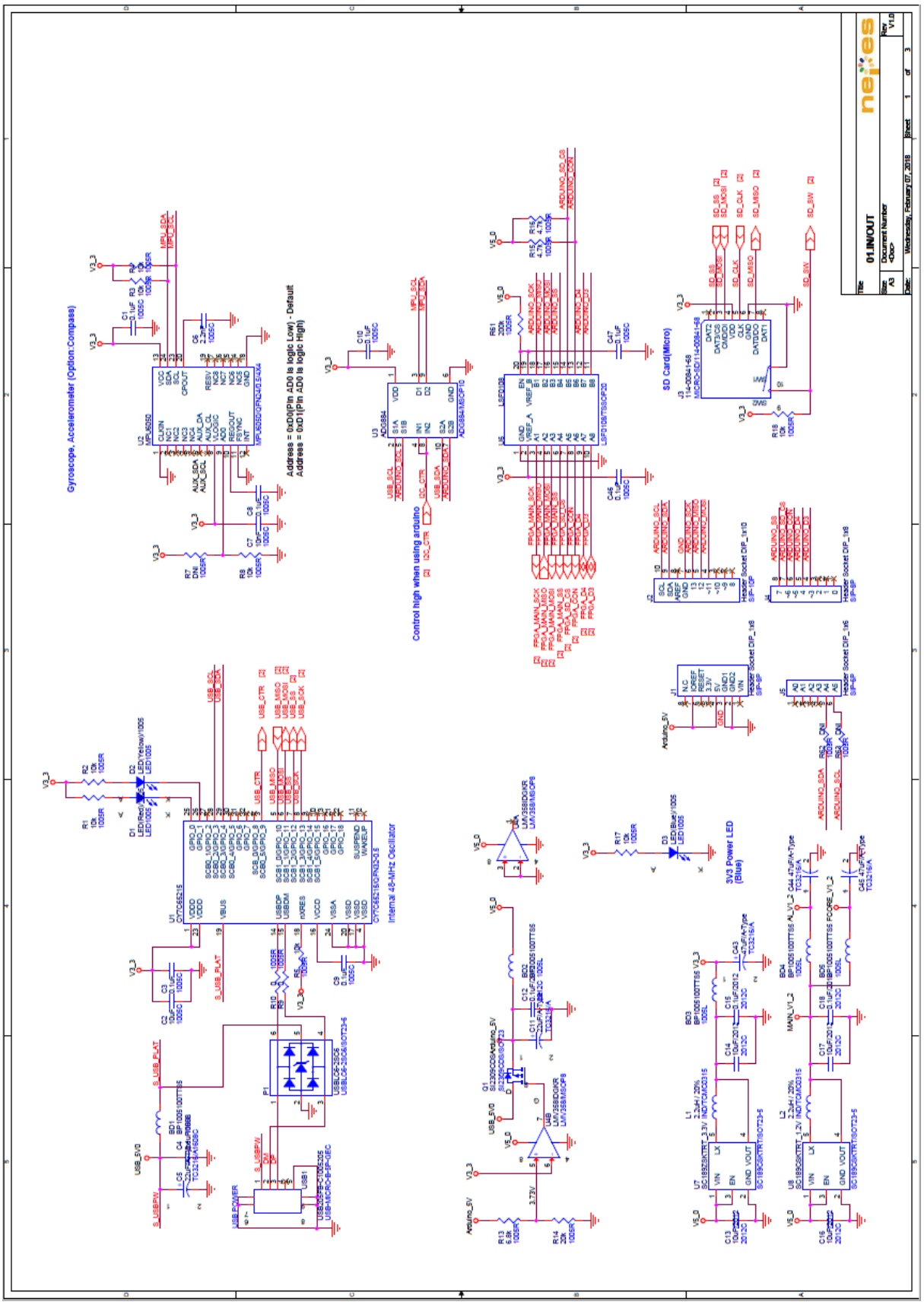
8 Troubleshooting

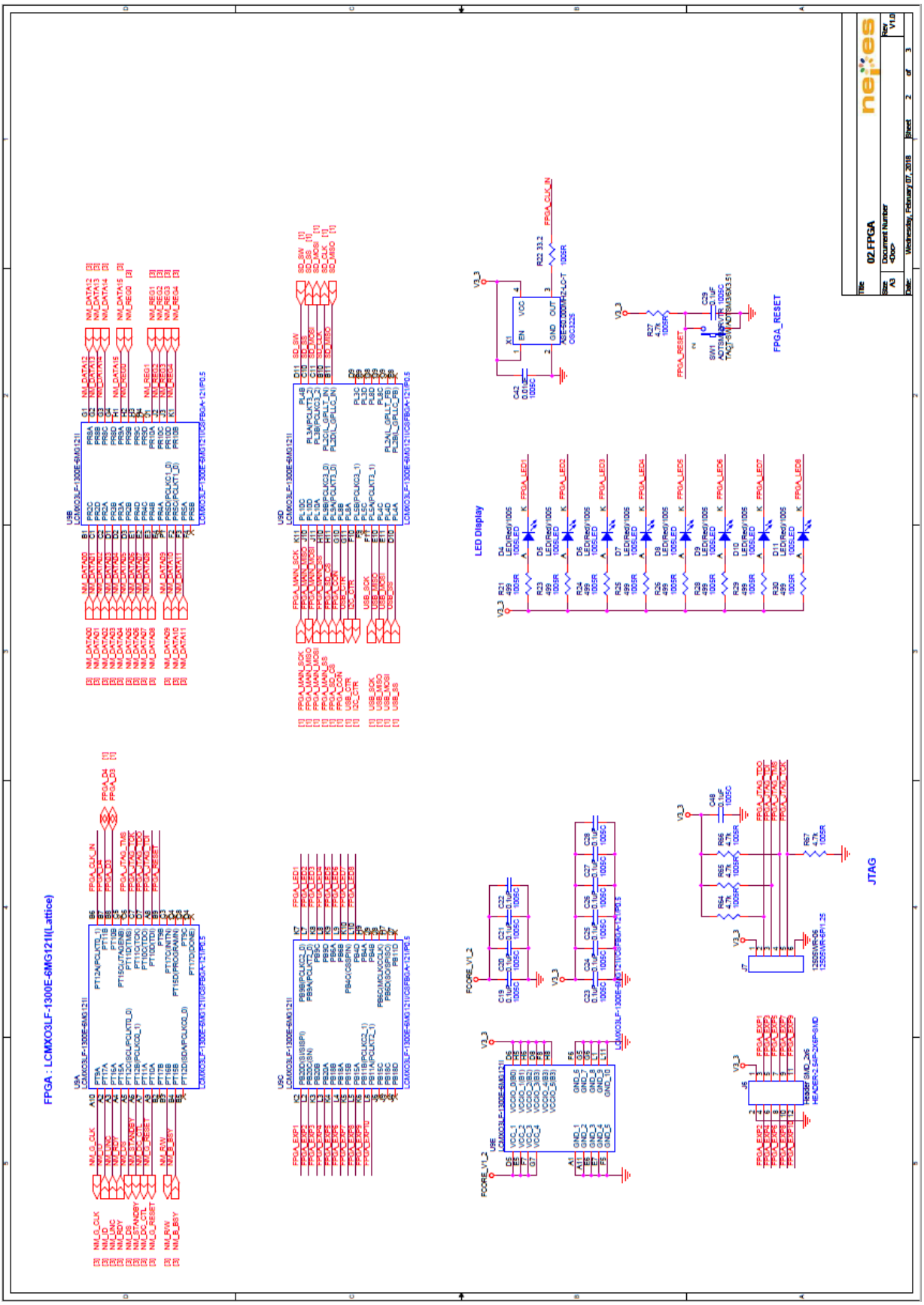
If NeuroShield is not responding to SPI communications

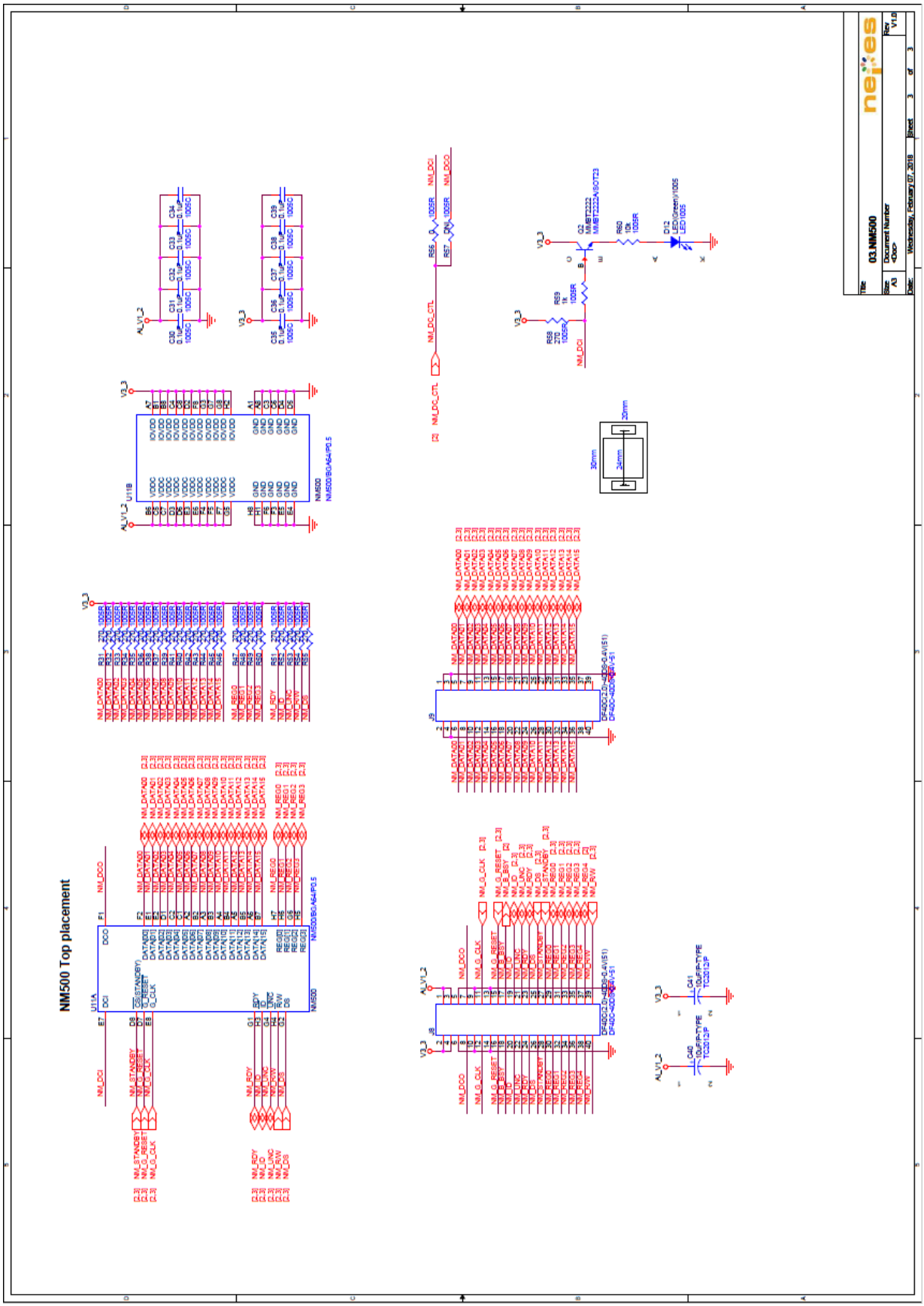
- Check the POWER LED, Verify that it is properly powered up (the blue LED should be lit)
- Push the reset button on the NeuroShield board and the main platform board
- Test the device with a different SPI speed (< 3Mhz)
- On the Arduino board:
    - Verify that the SPI Chip Select pins D5 to D7 on the Arduino connector are not used by another shield
    - Re-run the pre-provided for example code in this kit (NM500_SPI_Test.ino)
    - Verify that the pins D11 to D13 on the Arduino connector are reserved for the SPI communication in the NeuroShield and not assigned anywhere else in the code
- On the PC platform double check the:
    - Driver for the USB
    - USB port on the users PC
    - Cypress USB setting for the NeuroShield
- Finally, try re-running the pre-provided for example code in this kit (NM500_USBtoSPI.exe)

9 APPENDIX

NeuroShield Schematics




FPGA : LCMXO3LF-1300E-6MG121I(Lattice)
LED Display
JTAG
FPGA_RESET
02.FPGA
nexes
Document Number
Wednesday, February 07, 2018


NM500 Top placement
03.NM500
Document Number
<Doc>
Rev
V1.0
Size:
A3
Date:
Wednesday, February 07, 2018
Sheet
3
of
3

10 Revision History

Date	Version	Section	Change Summary
Feb 2018	1.0.5	9. APPENDIX	Update a schematic
Aug 2017	1.0.4	5. NeuroShield Description	Add contents Precautions when connecting the hardware
Aug 2017	1.0.3	3. Function Libraries 4. Example Code	Updated function libraries and Example code
Aug 2017	1.0.2	5. NeuroShield Description	Updated a picture of block diagram and hardware description
		6. SPI protocol	Updated a register map
Jul 2017	1.0.1	1. Introduction 2. Getting Started	General updated