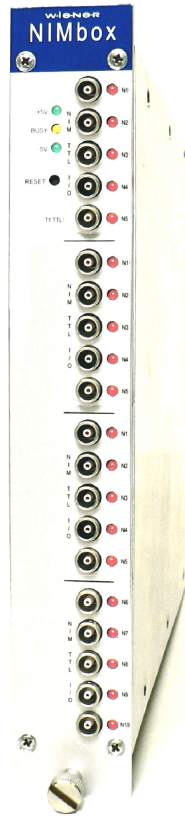


W-Ie-Ne-R NIMBOX NPN20



Nembox Series NIM Version – NPN20

User's Manual

General Remarks

The only purpose of this manual is a description of the product. It must not be interpreted as a declaration of conformity for this product including the product and software.

W-IE-Ne-R revises this product and manual without notice. Differences between the description in manual and the product are possible.

W-IE-Ne-R excludes completely any liability for loss of profits, loss of business, loss of use or data, interrupt of business, or for indirect, special incidental, or consequential damages of any kind, even if **W-IE-Ne-R** has been advised of the possibility of such damages arising from any defect or error in this manual or product.

Any use of the product which may influence health of human beings requires the express written permission of **W-IE-Ne-R**.

Products mentioned in this manual are mentioned for identification purposes only. Product names appearing in this manual may or may not be registered trademarks or copyrights of their respective companies.

No part of this product, including the product and the software may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the express written permission of **W-IE-Ne-R**.

Table of contents:

1	<i>Quick startup</i>	1
1.1	Power up.....	1
1.2	Hardware detection and software installation	1
1.2.1	Windows.....	1
1.2.2	Linux	1
1.3	Programming the FPGA	1
2	<i>Hardware description</i>	2
2.1	FPGAs	2
2.2	General description of Nembox	2
2.3	I/O submodules.....	2
2.4	SU704 – NIM I/O.....	3
2.5	NPN20 hardware configuration.....	4
3	<i>Software description</i>	5
3.1	USB communication	5
3.2	Addressing.....	6
3.3	Defining FPGA connections	7
3.3.1	Reserved values:.....	7
3.4	NIMbox/Nembox Port Mapping (Front Panel).....	7
3.5	Logic Pool for Labview™	8
3.5.1	DIO.....	9
3.5.2	Logic.....	10
3.5.3	Counter.....	10
3.5.4	LED	11

1 Quick startup

1.1 Power up

On power up, the power LEDs are active and, if the firmware has been installed, the “BUSY” LED will blink shortly. Nembox is shipped with preinstalled firmware which is consistent with the hardware version and with the position of the installed submodules on the main FPGA board. Immediately after the first power up the USB communication is possible but no configuration setup is loaded, unless the user saves a configuration in the internal EEPROM.

1.2 Hardware detection and software installation

1.2.1 Windows

Once Nembox has been powered and connected to a USB port, the hardware installation wizard will guide the user through the installation steps. It is recommended not to let Windows look for a proper driver. Choose instead to manually install the driver software from the Nembox CD.

If the CDROM drive is D, the driver directory is

```
D:\USB-Driver-FX2\
```

It is recommended to save the complete directory to the harddisk in the LabVIEW directory for later reinstalling and for availability of the interface DLL “USBBoxLib.dll”, which is also referred by LabVIEW.

After successful installation the user can verify the proper operation of Nembox in the Control Panel: in the Device Manager, there should be a new entry named DL7XX LogicBox.

Alternatively, the user can start the LabVIEW™ based program NemboxTest, available in the CD, which will ask for the Nembox USB address string and will return the Nembox device ID.

IMPORTANT: the USB address string, that must be used to identify Nembox under LabVIEW™, can be found on a label on the backside of the module.

1.2.2 Linux

Please check www.wiener-d.com for updates.

1.3 Programming the FPGA

In order to quickly program the FPGA with predefined functionalities, it is recommended to use LabVIEW™. Copy the Nembox Logic Pool VIs from the NIMbox/Nembox CD to the VI user library directory of LabVIEW™.

For example:

```
copy from: D:\Labview 8.2\LogicPool to:  
C:\Programs\National Instruments\LabVIEW 8.2\user.lib
```

Tests and Demos can be copied to an arbitrary directory.

By starting LabVIEW™ the new Logic Pool VIs will be available. These VIs are described in this manual. By connecting the Logic Pool VIs and by starting your application, you will load a configuration to the FPGA. This configuration will hold as long as you don't overwrite it by starting another application or as long as you don't reset or switch off Nembox. The “close” VI will ask the user whether he wants to permanently save the configuration on the internal EEPROM so that it won't be lost in case of power off..

When starting VIs for the first time it may be necessary to assist LabVIEW manually in searching the path to USBBox.dll.

2 Hardware description

2.1 FPGAs

A field-programmable gate array is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories.

An hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer to implement any logical function—hence field-programmable. However, this hardware programming is somewhat complex and requires programming tools (for example VHDL) and considerable development effort. For this reason, Nembox is shipped with a special software, Logic Pool, that provides the user with a large number of preprogrammed function modules that are typically useful in signal processing.

2.2 General description of Nembox

Nembox is a programmable NIM module based on a FPGA board (DL706) with 4 slots for I/O submodules that serve as interface between the FPGA I/O signals and the signals in the external environment. It is equipped with a USB port for programming and read out and a connector for direct FPGA programming/debugging.

Its 100 MHz clock makes Nembox well suited for processing signals with length down to 10 ns and frequencies of several MHz. Such signals are common in nuclear and particle physics applications, where NIM and TTL standards are used for signal transmission and processing.

The NPN20 version can be used as 20 channel programmable NIM or TTL logic unit or scaler (or both), for implementing, for example, triggers and gate generation. By default, the programmable input/output ports are set to NIM. They can be individually set to TTL by changing a jumper, therefore NIM<->TTL conversion is also possible. Internal preconfigured Logic modules and Counter modules enhance more complex applications.

2.3 I/O submodules

Presently, the following submodules are available for Nembox:

Table 1: Nembox I/O submodules.

Submodule	Status (10.06)	Function
SU700	Available	5x TTL I/O – LEMO COAX
SU701	Hardw. avail.	16x TTL I/O
SU703	Available	4x Discriminator and 1x TTL I/O – LEMO COAX
SU704	Available	5x NIM/TTL I/O – LEMO COAX
SU705	Hardw. avail.	16 MByte RAM
SU706	Available	1x ADC (100 Mhz) ,2x TTL I/O – LEMO COAX
SU707	Hardw. avail.	8 x LVDS I/O
SU709	Hardw. avail.	8 x Temperaturesensor
SU710	Available	2x Fast DAC (100 Mhz)
SU711	Hardw. avail.	6 x programmable Delayline 0,5ns .. 128 ns
SU712	Not yet avail.	16 x ADC (5 us, 14 Bit)

SU713	Not yet avail.	16 x DAC (14 Bit)
-------	----------------	-------------------

NIMbox/Nembox NPN20 is outfitted with 4 SU704 submodules.

2.4 SU704 – NIM I/O

SU704 has 5 identical LEMO COAX I/O connectors to be used as programmable digital I/O ports. Every I/O port supports both NIM and TTL levels, but the selected level is defined by a jumper setting on the SU704 board (default level is NIM). Input impedance can be set to 50 Ohm through a relais.

Every NIM output can draw a current of -16mA, thus with a 50 Ohm impedance the level is -0,8V. The corresponding input threshold, with 50 Ohm set, is -0,4V. Maximum frequency is 100 MHz and propagation delay less than 4 ns.

Table 2: SU704 (NIM I/O) pin assignments.

Pin N.	Function
1	+ 5V
2	+5V
3	INTTL5 (TTL in, ch. 5)
4	OUT5 (NIM/TTL out, ch. 5)
5	INECL5 (NIM in, ch. 5)
6	ENA5 (enable TTL out, ch. 5, low active)
7	LED5
8	REL5 (relais 5)
9	INTTL4 (TTL in, ch. 4)
10	OUT4 (NIM/TTL out, ch. 4)
11	INECL4 (NIM in, ch. 4)
12	ENA4 (enable TTL out, ch. 4, low active)
13	LED4
14	REL4 (relais 4)
15	INTTL3 (TTL in, ch. 3)
16	OUT3 (NIM/TTL out, ch. 3)
17	INECL3 (NIM in, ch. 3)
18	ENA3 (enable TTL out, ch. 3, low active)
19	LED3
20	REL3 (relais 3)
21	INTTL2 (TTL in, ch. 2)
22	OUT2 (NIM/TTL out, ch. 2)
23	INECL2 (NIM in, ch. 2)
24	ENA2 (enable TTL out, ch. 2, low active)
25	LED2
26	REL2 (relais 2)
27	INTTL1 (TTL in, ch. 1)
28	OUT1 (NIM/TTL out, ch. 1)
29	INECL1 (NIM in, ch. 1)
30	ENA1 (enable TTL out, ch. 1, low active)
31	LED1
32	REL1 (relais 1)
33	-
34	-
35	GND
36	GND

2.5 NPN20 hardware configuration

The slots for submodules in the main FPGA board are named MOD0, MOD1, MOD2 and MOD3 counting top to bottom. In the NPN20 version all slots are occupied by SU704 (NIM I/O) units.

Please note that the delivered software depends on this hardware configuration, therefore only advanced users should consider removing, substituting or swapping submodules, because this operation requires firmware reprogramming.

3 Software description

3.1 USB communication

The Nembox USB interface provides a number of commands for communicating with the system or with the submodules. All commands are „byte oriented“ and identified by an ASCII character. Addresses and data (1 to 4 bytes) are binary and the sequence of word and longword transfers is big endian (most significant bytes first).

Table 3: System.

Command	Send	Receive	Function
#		4 Bytes	Send ID 31..0
R			System Reset

Table 4: Transfer.

Command	Send	Receive	Function
A	4 bytes		set address pointer A31..0
E	3 bytes		set address pointer A23..0
M	2 bytes		set address pointer A15..0
S	1 byte		set address pointer A7..0
A		4 bytes	read address pointer A31..0
+			increase address pointer A31..0 by 1
-			lower address pointer A31..0 by 1
N	2 bytes		set counter N15..0 for block transfer (A autoincr)
F	2 bytes		set counter N15..0 for FIFO transfer (A)
L	N*4 bytes		write N longword(s)(A) (*)
L		N*4 bytes	read N longword(s)(A) (*)
T	N* 3 bytes		write N Triple(s)(A) (*)
T		N* 3 Bytes	read N Triple(s)(A) (*)
W	N* 2 Bytes		write N Word(s)(A) (*)
W		N* 2 Bytes	read N Word(s)(A) (*)
B	N* 1 Byte		write N Byte(s)(A) (*)
B		N* 1 Byte	read N Byte(s)(A) (*)

(*) Block transfers:

In case the block transfer counter N (commands ‚N‘ and ‚F‘) is set to 1, a corresponding 1 byte transfer according to the selected datawidth is executed.

When $N > 1$, N read or write cycles are executed and in case the counter was set with the `,N'` command, the address pointer is incremented at each transfer. In case the command `,F'` is used, all transfers are executed at the same address (typ. FIFO).

The counter N after a block transfer is always set to 1, the address pointer A is resetted to the start address.

CRC Check: (not yet available. Status: 14.6.2007)

Command	Send	Receive	Function
?		B0	send CRC checksum of all bytes sent
!		B0	send CRC checksum of all bytes received

Example sequence (Number format in HEX; ','=ASCII):

Sent	Received	Function
,#'	00000100	ID number of the module=256
,A'00000001		set address pointer A=1
,S'0A		set address pointer A=10
,a'	0000000A	read address pointer
,L'00000200		write to current address (A=10) the 32Bit value=512
,B'FF		write the byte value=255 to the address A=10
,w'	02FF	read the value 767 from address A=10
,+'		increase the address counter by 1 (A=11)
,W'0304		write the value 772 to A=11
,N'0002		set block transfer counter (Autoincr) to 2
,-'		set address counter back to A=10
,b'	FF04	block transfer of 2 bytes with auto increment

3.2 Addressing

All Logic Pool modules are addressed through a 24 bit address:

Address bits	Value range	Function
A31..A24		irrelevant
A23..A16	0..255	type address of a Logic Pool module (e.g. „T“, „L“,...)
A15..A8	1..255	Module address of a Logic Pool module
A7..A0	0..255	sub address for arbitrary parameters

Every module must notify his presence to subaddress=0 with a byte 0..254. If no module is present at a specific address, the value 255 (or FF) is returned. Therefore an application (e.g.

OPEN in LabVIEW) can check that all modules are available and create a table of modules and addresses.

All other module parameters on further sub addresses are specific for each module. The width of the data word is arbitrary, between 1 and 4 bytes.

3.3 Defining FPGA connections

Each module has typically one or more signal inputs and outputs.

Every output (**MUX_**) has a specific connection value, that can be read out by every module. Moreover, the logic state (**STATE_**) at output can be inquired at any time. Usually these values are stored on the read-subaddresses 0..n .

Every input (**_MUX**) is outfitted with a multiplexer switch, that allows to set up the connection to any output. Therefore any arbitrary interconnection between modules becomes possible. The multiplexer are set to the user defined connection through a value in a register. Typically, the input multiplexer values are written from the user to the write-subaddresses 0..n .

Example: the output of the module T10 should be directed to the input of module I3

The command ,E''T'0A00'b' delivers the connection value = 02.

Hence, the command ,E''I'0300'B'02 connect the signal as intended.

3.3.1 Reserved values:

By setting the MSB (most significant bit) of the multiplexer register, it is possible to invert every input without using other resources (z.B.: ,E''I'0300'B'82).

- Value **0** characterizes an open (not connected) input or output.
- Value **127** produces a short trigger pulse, after that it is set to 128 (not yet implemented).
- Value **128** corresponds to a constant LOW level.
- Value **255** corresponds to a constant HIGH level.

3.4 NIMbox/Nembox Port Mapping (Front Panel)

The 20 I/O ports of NIMbox/Nembox NPN20 have a default assignment as follows:

Connector	Led
N1	I1
N2	I2
N3	I3
N4	I4
N5	I5
N6	I6
N7	I7
N8	I8
N9	I9

N10	I10
N11	I11
N12	I12
N13	I13
N14	I14
N15	I15
N16	I16
N17	I17
N18	I18
N19	I19
N20	I20

Table 5: Mapping of physical I/O ports on front panel

“N” is a programmable NIM or TTL I/O port and “I” a LED.

This labeling is very important in order to map the physical position of the I/O ports to the corresponding software identifiers. While using LabVIEW™ VIs, the integer number which is part of the port label should be provided as input.

3.5 Logic Pool for Labview™

Logic Pool is a set of tools for programming the NIMbox/Nembox FPGA under Labview.

It consists of the following components:

- Firmware
- USB communication VI
- Applications (VIs)

The firmware is preinstalled on NIMbox/Nembox and can only be modified by advanced users with detailed hardware information. By default NPN20 is equipped with the following FPGA resources:

- 20 x NIM/TTL I/O, configurable as NIM or TTL I/O
- 8 x Logic
- 4 x Counter

These resources are described in this chapter and each one has its corresponding application VI for easy access.

The USB communication VIs is a VI interface to the file USBBoxLib.dll

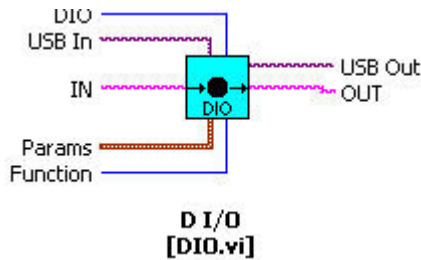
The application VIs implement the following functionalities:

- DIO to use NIM/TTL I/O resources
- DISCRIMINATOR (not present in the NPN20 version)
- LOGIC
- COUNTER
- LED
- TDC (not present in the NPN20 version)

- ADC, DAC (not present in the NPN20 version)

In order to make it possible to use more than one NIMbox/Nembox instance at one, all VIs have the I/O parameters “USB In” and “USB Out” that can be used to distinguish sessions. By using the provided OPEN.vi instrument, the selected USB device will be stored in a local variable for reference of all other modules. This simplifies wiring of the moduls for single Nembox setups.

3.5.1 DIO



Picture 1: Digital I/O Virtual Instrument

This VI controls a physical NIM or TTL I/O port on NIMbox/Nembox.

“Function” can be set to

- Connect (default and normally executed once initially)
- Set IN (sets the output to True or False, overrides any connection to OUT)
- Get OUT (returns the status of the port)

The “Params” cluster (provided by the user) is used to pass values to “Function”. For example, it can be used to determine the level of the I/O port. This could be chosen between:

- TTL High Imp.
- TTL 50 Ohm
- NIM Open
- NIM 50 Ohm
- A debounce value may cancel spurious or intermittent input signals

The “DIO” integer value (provided by the user) identifies the physical port according to Table 5.

For example, by choosing a TTL option for “Params” and the number 3 for “DIO”, the resulting physical port will be the T3, i.e. the 11th (where the 1st is the uppermost and the 20th the lowermost port).

The VI has an input (OUT, because it is used to feed the real output signal which will actually be generated by Nembox) and an output (IN, because it is used to receive an input signal into the Nembox).

DIO:

,T'n	Bytes	Read	Bytes	Write
0	1	MUX_IN	1	OUT_MUX
1	1	STATE_IN	1	Debounce

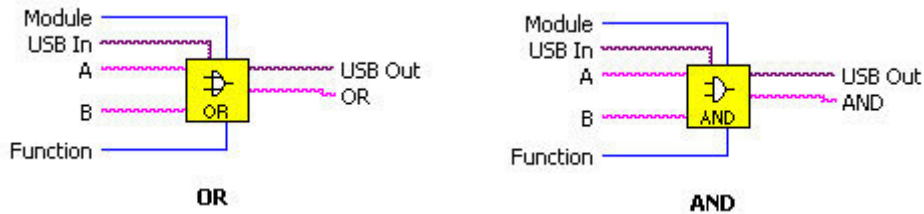
2	1		1	Mode: termination (0), NIM(1)
---	---	--	---	-------------------------------

Debounce:

=0: debouncing disabled

=1..255: debouncing with 1..255 ms enabled

3.5.2 Logic



This set of VIs provides the basic logic functions.

LOGIC:

,L'n	Bytes	Read	Bytes	Write
0	1	MUX_Out	1	A_MUX
1	1	STATE_Out	1	B_MUX
2			1	Mode
3			1	FF

Mode:

0: OR

1: AND

2: XOR

3: RS-FF

4: S-FF

5: D-FF

6: Differentiator

7: Differentiator (Start Asynchronously)

8: Synchronisator

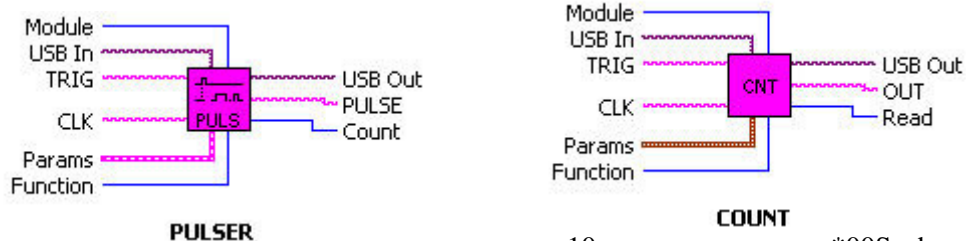
9: Synchronisator (Start Asynchronously)

FF (Flip Flop):

0: FF is deleted

1..255: FF is set

3.5.3 Counter



This set of VIs can be used for counting (scalers) or to generate an arbitrary train of pulses.

COUNTER:

,C'n	Bytes	Read	Bytes	Write
0	1	MUX_Out	1	A_MUX
1	1	STATE_Out	1	B_MUX
2	1..4	Counter	1	Mode, Clear Counter
3			1..4	Pulse marks, Clear Counter
..n			1..4	Pulse marks, Clear Counter

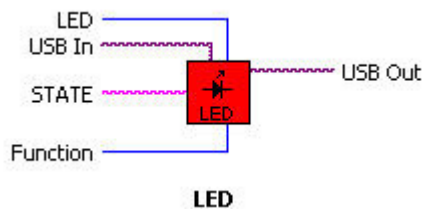
Mode:

0: COUNTER

1: PULSER

2: ASYNC PULSER

3.5.4 LED



This VI is used to control the status of the LEDs. The integer "LED" determines which LED is being programmed, according to Table 5.

LED:

,I'n	Bytes	Read	Bytes	Write
0	1	0	1	LED_MUX

