Open Communication

Standard labZY FPGA Designs

Revision 7.1

labZY Standard Firmware

Revision 3.0

Contents

Overview

Applicable Devices

This document applies to the following labZY Tools:

nanoMCA

nanoMCA-SP

nanoXRS

nanoDPP

"**labZY Tool**" refers to any of these devices.

Computer Connections

labZY Tools are connected to a computer using one of the following hardware connections: direct USB cable, Ethernet connection through nanoETH module and wireless connection using the nanoBT interface module. When the driver of each one of these connections is installed a virtual COM port is created. Fig. 1 shows the PORT selection dialog of the labZY-MCA software. The selection list include all available virtual COM ports of the labZY Tools.

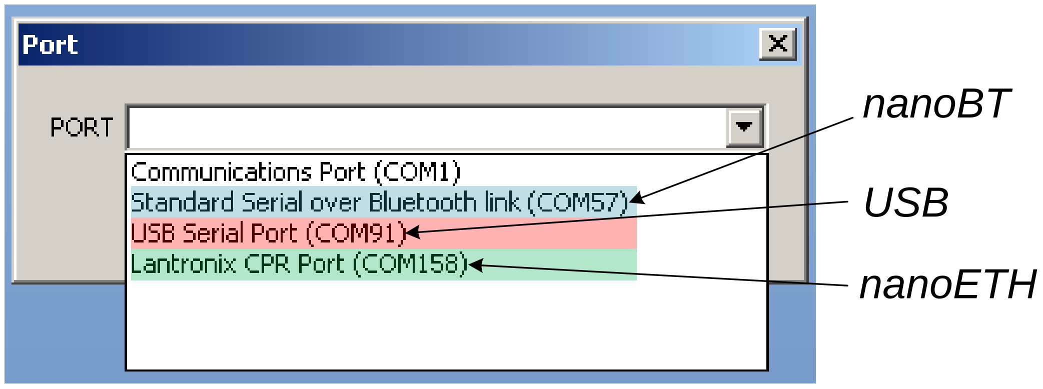

Port
PORT
Communications Port (COM1)
Standard Serial over Bluetooth link (COM57)
USB Serial Port (COM91)
Lantronix CPR Port (COM158)
nanoBT
USB
nanoETH

Fig. 1 PORT selection dialog of labZY-MCA software.

The virtual COM ports emulate standard serial ports and must be configured before being used. The RTS functionality is built-in feature of the nanoBT and nanoRTH modules.

The USB interface cable may provide power to the labZY eliminating the need of an external power supply. The external power supply will be required for the Bluetooth and the Ethernet connections. The external power supply can be used as a backup power source when the labZY Tools are connected through the direct USB interface. The USB interface cable and the interface modules are connected to the IO port of the labZY Tools (Fig. 2). The external power is applied through a standard Mini B USB cable to the PWR port of the labZY Tools.

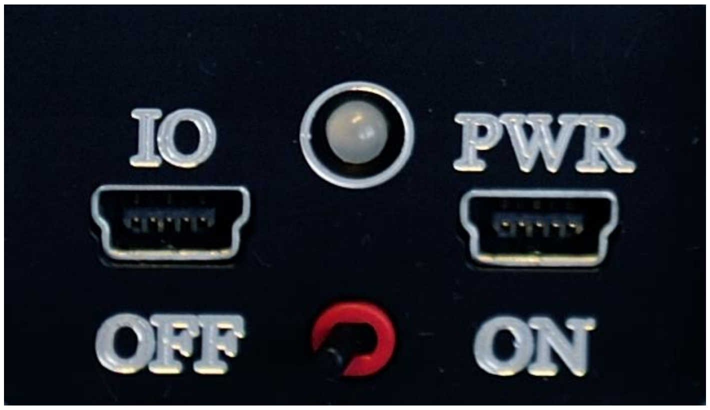


Fig. 2 Interface (IO) and power (PWR) connectors of the labZY Tools.

UART Interface

labZY Tools can be interfaced to a microcontroller or a microprocessor based system using an UART port with 3.3V CMOS logic levels. The hardware connection is through a 6 inch (15cm) USB Mini B cable with flying leads shown in Fig. 3.

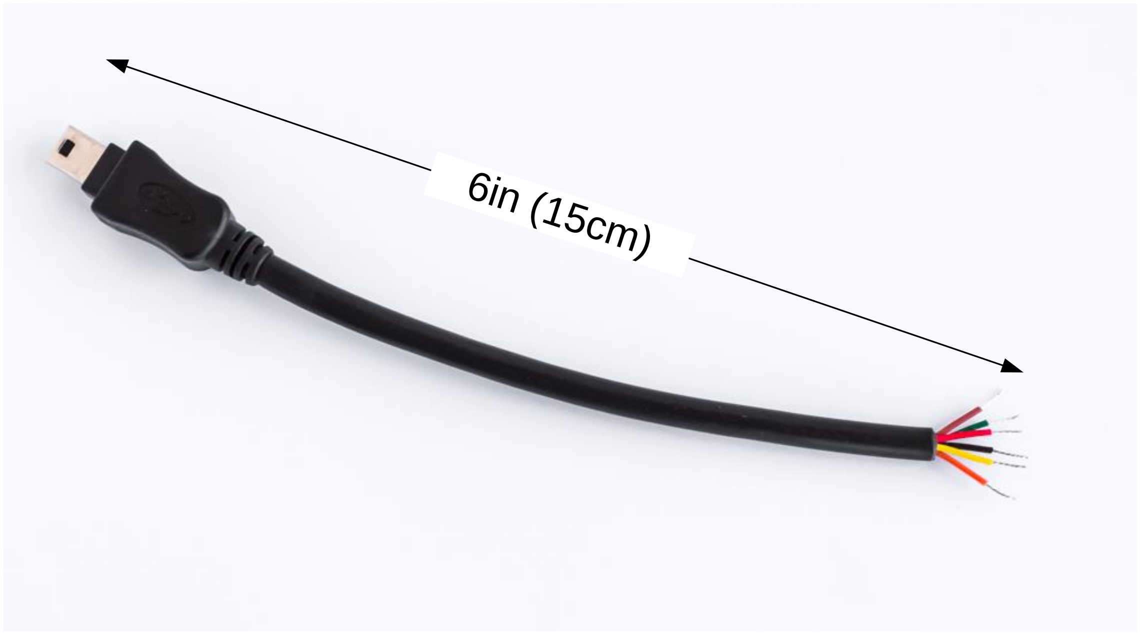


Fig 3. Cable for connecting the labZY Tools directly to UART ports.

The physical connection between the labZY Tool and the UART port of a microcontroller is shown in Fig. 4. The labZY tool can provide +3.3V power and up to 300mA current to the microcontroller or other external circuits. The +3.3V is available on the red lead of the interface cable. If this power is not needed the red lead should be isolated and kept unconnected. Note that the UART interface requires that a +5V external power is supplied to the PWR connector of the labZY Tool.

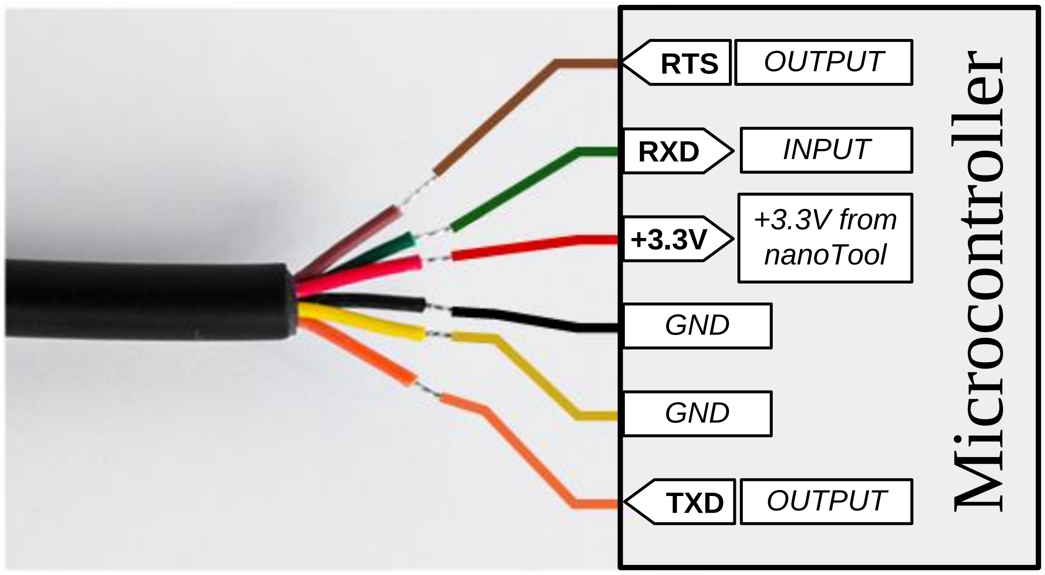


Fig. 4 Microcontroller connection to labZY tools using UART interface.

The output line of the microcontroller serial TXD line is connected to the orange lead of the interface cable. The serial input line is connected to the green wire. The microcontroller flow control line RTS is connected to the brown lead. An example of the UART signal waveforms is shown in Fig. 5. Note that the asserted level of the RTS line is logic LOW (3.3V CMOS).

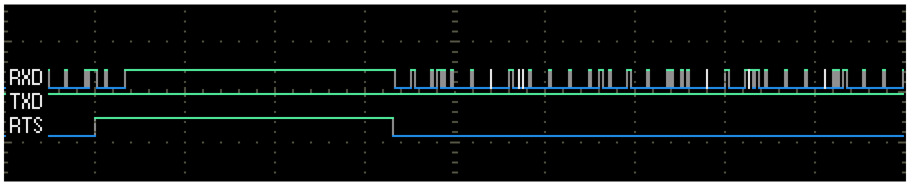


Fig. 5 UART interface signals.

The RTS line is used to control the data flow from the labZY Tool into the microcontroller and to prevent loss of data. The microcontroller must de-assert RTS line in advance of its receiving buffer becomes full leaving room to receive at least two extra bytes from the labZY Tool. This is an important consideration as there might still be data on the line and/or in the labZY Tool transmit register which has to be received even after RTS has been de-asserted. When the microcontroller is ready to accept more data it should assert the RTS line and keep it asserted until the receiving buffer gets close to become full. The labZY Tool serial port will stop transmission of data as soon as it senses the asserted state of the RTS signal and will pause the transmission until RTS is again asserted. The labZY tool can accept all data send by the microcontroller without any interruption of the transmission. Therefore, there is no need of CTS hardware control to the microcontroller. If the microcontroller UART port is equipped with a CTS line then it should be asserted all the time. The timing diagram in Fig. 6 illustrates the data transmission using the RTS control line showing the extra bytes transmitted after the RTS line is de-asserted.

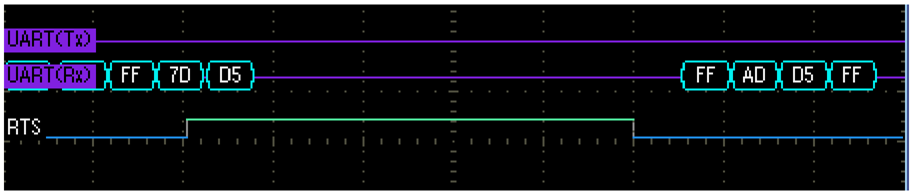


Fig. 6 Pausing data flow using the RTS line.

Communication Protocol

Port Settings

The settings for computer virtual com ports or the UART are as follows:

Baud Rate: 460800

Data Bits: 8

Stop Bits: 1

Parity: N

Handshaking: Hardware, RTS only,

UART: RTS asserted state is active LOW

Commands and Responses

The communication between a labZY Tool and the host (computer or microcontroller) is accomplished using commands and responses sent in half-duplex serial transmission as shown in Fig. 7.

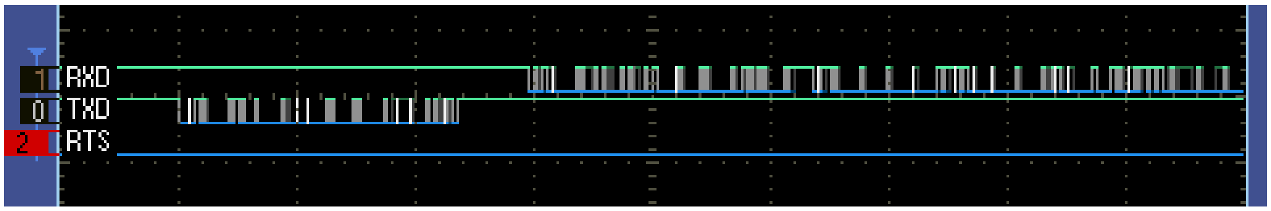


Fig. 7 Half duplex serial communication. Shown are the host signals.

The host first sends a command through its TXD line and then listens for a response on its RXD line. In normal operation there should be no simultaneous transmissions on the TXD and the RXD lines. The labZY Tool is a slave device while the host is a master device. Therefore, the master initiates the communication by sending a command. Subsequent commands are sent by the host only after a complete and valid response has been received from the labZY Tool or a time-out interval has been reached. The time-out interval should be equal or greater than five seconds.

There are only two commands to access data and to control the labZY Tool. The FPGA spectral data (spectrum) and the FPGA registers are accessed as memory data at specific addresses. Each memory location is organized and accessed as a 16-bit word. Each 16-bit word consists of two bytes and two words comprise a long word. Bytes representing words and long words, and words representing long words are stored in the memory in a Little-Endian order.

The host sends and receives the data as a sequence of bytes. The first byte in the sequence is designated as zero byte (BYTE[0]) followed by bytes whose order is incrementally numbered. Bytes are sent in order that follows the Little-Endian rule when the bytes represent words or long words.

The last byte in each command and in the corresponding response from the labZY Tool is a check sum byte - **CHKS**. The content of the check sum byte is obtained in two steps. First, a byte sum is calculated by adding all bytes in the command/response excluding the check sum byte. In the byte sum calculation the bytes are treated as unsigned numbers and the overflow is ignored. This sum is stored in the check sum byte. That is, the check sum byte acts as an accumulator of all bytes of the command/response. Secondly, all bits of the obtained byte sum are inverted and the result is incremented by 2 ignoring the overflow. This value is the check sum **CHKS** included as a last byte in the host command or the labZY Tool response.

The data is read or written in words starting at an initial address **IADR** or **IADW** respectively. This address is specified in the commands sent by the host to the labZY Tool. An **AutoIncrement** bit is a part of the address information send by the host. If the **AutoIncrement** bit in the address field of the command is set to one, then the data address will be incremented automatically after each word reading/writing. The start address is the initial address specified by the host. When reading data and the **AutoIncrement** bit is zero then the data will be read from the same initial address if the number of words to be read is greater than one. When more than one word is to be written with the AutoIncrement bit set to zero then the data at the specified address will be overwritten until the last data word send by the host is written. The host commands also include the number of bytes to be read or written to the FPGA memory - **TNBR** or **TNBW** respectively.

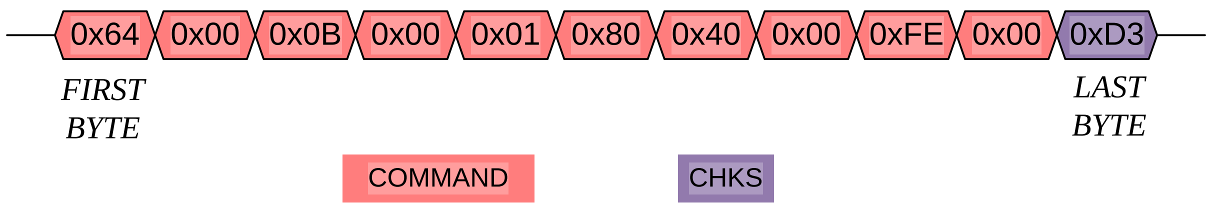
READ COMMAND

1. Host Read Command

BYTE[0,1]:WORD[0] = **100**	*- Command code.*
BYTE[2,3]:WORD[1] = **11**	*- Number of bytes in the read command including the check sum byte.*
BYTE[4,5,6,7]:LONG WORD[1] = BIT[21..0], BIT22, BIT[31..23]	
BIT[21..0]] = **IADR**	*- FPGA Address of the first data word to be read.*
BIT22 = **AutoIncrement**	*- If AutoIncrement=1, FPGA word address automatically increments as words are read beginning with IADR, if AutoIncrement=0 the word at address IADR is read TNBR times.*
BIT[31..23] = **0**	
BYTE[8,9]:WORD[4] = **TNBR**	*- The number of FPGA data bytes to be read (twice the number of FPGA data words to be read). TNBR must be an even number.*
BYTE[10] = **CHKS**	*- Check sum byte.*

Example: Reading 127 FPGA Registers starting from Register[1] at FPGA address 0x8001. Bytes are sent in Little-Endian order.

WORD[0]	WORD[1]	LONG WORD[1]	WORD[4]	CHKS
0x0064	0x000B	0x00408001	0x00FE	0xD3



Example Check Sum calculation:

$\mathrm{ByteSum} = \sum_{k=0}^{9} \mathrm{BYTE[k]}, \quad \mathrm{ByteSum} = 0\mathrm{x2E}, \quad \sim\mathrm{ByteSum} = 0\mathrm{xD1},$

CHKS = 0xD1 + 0x02 = 0xD3

2. *labZY Tool Response to Host Read Command*

BYTE[0,1]:WORD[0] = **100** - *Response Code (same as the Read Command Code).*

BYTE[2,3]:WORD[1] = **25 + TNBR** - *Number of bytes in the response including the check sum byte.*

BYTE[4,5,6,7]:LONG WORD[1] = BIT[21..0], BIT22, BIT[31..23]

BIT[21..0] = **IADR** - *FPGA Address of the first data word to be read (same as in the command).*

BIT22 = **AutoIncrement** - *AutoIncrement bit (same as in the read command).*

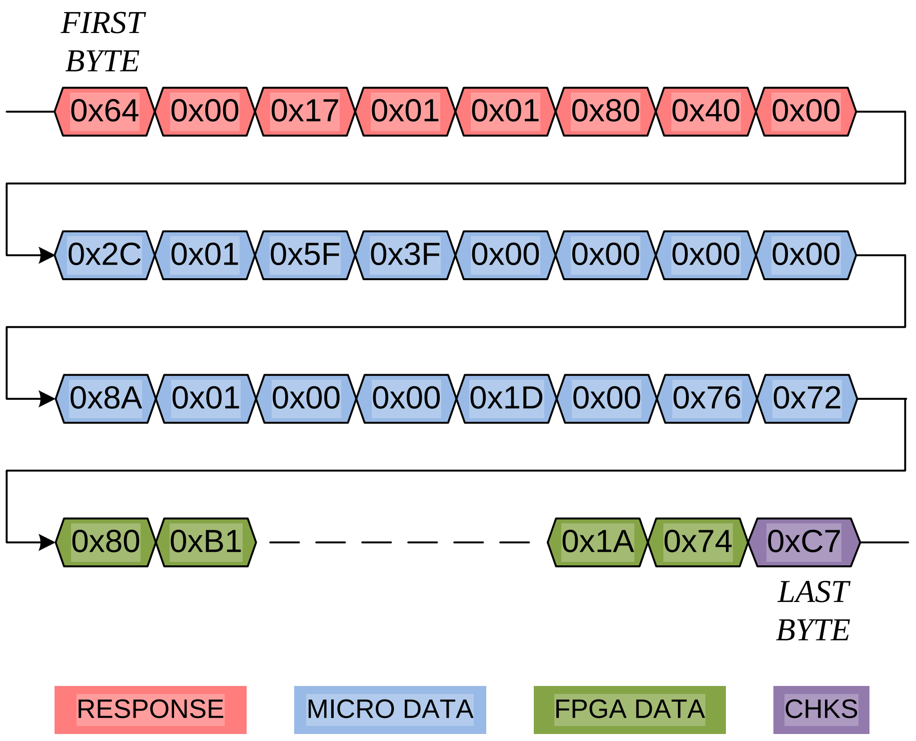
BIT[31..23] = **0**

BYTE[8..23]: WORD[4..11] = **MICRO Data** - *Hardware status data from the labZY Tool microcontroller, read only or read volatile.*

BYTE[24..23+**TNBR**]:WORD[12..11+**TNBR/2**] = **FPGA DATA**

BYTE[24+**TNBR**] = **CHKS** - *Check sum byte.*

Example: labZY Tool response to the host command to read 127 FPGA Registers starting from Register[1] at FPGA address 0x8001. Note that the response always includes the MICRO Data along with the Response and the FPGA Data. Bytes are received in Little-Endian order.



WRITE COMMAND

1. *Host Write Command*

BYTE[0,1]:WORD[0] = **110**	*- Command code.*
BYTE[2,3]:WORD[1] = **9 + TNBW**	*- Number of bytes in the response including the check sum byte. TNBW is the number of FPGA Data bytes to be written. TNBW is always an even number. NOTE: The maximum number of FPGA data bytes in each write command is 512.*

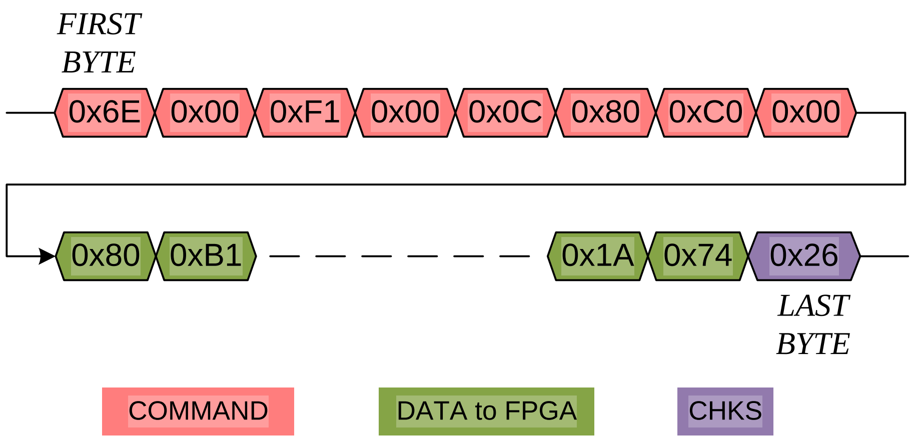
BYTE[4,5,6,7]:LONG WORD[1] = BIT[21..0], BIT22, BIT23, BIT[31..24]

BIT[21..0] = **IADW**	*- FPGA Address of the first data word to be written.*
BIT22 = **AutoIncrement**	*- If AutoIncrement=1, FPGA word address automatically increments as words are written beginning with IADW, if AutoIncrement=0 the word at address IADW is overwritten TNBW times.*
BIT23 = **1**	
BIT[31..24]= **0**	

BYTE[8..7+**TNBW**]:WORD[4..3+**TNBW/2**] = **DATA to FPGA**

BYTE[8+**TNBW**] = **CHKS** *- Check sum byte.*

Example: Writing data to 116 FPGA Registers starting from Register[12] at FPGA address 0x800C with automatic address increment. Bytes are sent in Little-Endian order.



2. *labZY Tool Response to Host Write Command*

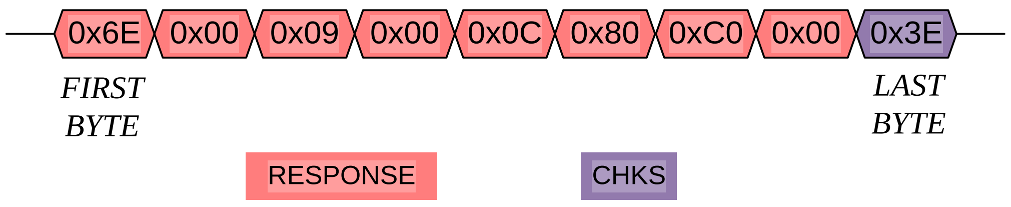
BYTE[0,1]:WORD[0] = **110**	*- Response Code (same as the Read Command Code).*
BYTE[2,3]:WORD[1] = **9**	*- Number of bytes in the response including the check sum byte.*

BYTE[4,5,6,7]:LONG WORD[1] = BIT[21..0], BIT22, BIT[31..23]

BIT[21..0] = **IADW**	*- FPGA Address of the first data word to be written (same as in the command).*
BIT22 = **AutoIncrement**	*- AutoIncrement bit (same as in the write command).*
BIT23 = **1**	
BIT[31..24]= **0**	

BYTE[8] = **CHKS**	*- Check sum byte.*

Example: labZY Tool response to a command to write data to 116 FPGA Registers starting from Register[12] at FPGA address 0x800C with automatic address increment. Bytes are received in Little-Endian order



The MICRO Data is a set of eight words that are transmitted as part of the labZY Tool response to the host READ COMMAND. WORD[4] (BYTE [8,9]) through WORD[11] (BYTE [22,23]) in the response represent the MICRO Data. The Micro Data is read only data and represent constant or volatile values. The volatile values may change between consecutive readings.

WORD [4] = Constant, Unsigned, Firmware Version *100. e.g. 321 = Version 3.21

WORD [5] = Constant, Unsigned, labZY Tool serial number.

WORD [6] = Volatile, Unsigned, nanoXRS only, Detector Bias Voltage [V];

Other tools RESERVED.

WORD [7] = RESERVED.

WORD [8] = Volatile, Signed, nanoXRS detector temperature;

Other tools Input D slow ADC reading 0 to 2500 [mV].

WORD [9] = Volatile, Unsigned, nanoXRS cooling power 0 to 330; Divide by 3.3 to find % power.

Other tools RESERVED.

WORD [10] = Volatile, Signed, labZY Tool internal temperature [C].

WORD [11] = RESERVED.

FPGA Data

FPGA Data is stored internally in the FPGA. The FPGA Data includes the Spectrum and the Registers. The Spectrum is stored in the FPGA internal memory and can be accessed as words (16bit) at word address 0x000 to 0x7FFF. Address space 0x8000 to 0x807F is used to access 16bit registers that control the hardware and provide status information.

Spectrum

The labZY Tools store spectral data in a spectrum with a fixed size of 16384 (2^{14}) channels. The counts in each channel are stored in a long word using the Little-Endian rule. Therefore, the content of each channel is accessed by reading two words (two memory locations of the memory address space). For instance, the counts of channel zero can be obtained by reading the words at addresses 0x0000 and 0x0001. The counts of the last channel are represented by the words at addresses 0x7FFE and 0x7FFF. With a single read command multiple channels can be read sequentially by setting the address AutoIncrement bit. It is recommended that few thousand channels are read by a single read command, e.g. 4096 channels.

Registers

Physically the registers are part of the FPGA designs supplied by labZY. Registers are used to control the operation of the labZY Tool with labZY provided FPGA functionality. The registers are also used to access data such as hardware information, acquisition time, noise measurements etc. From a software point of view these registers are 16-bit words. There are 128 registers located at addresses 0x8000 to 0x807F. The address 0x8000 is the base address. Registers are numbered sequentially from 0 to 127. The register number is the offset relative to the base address.

Application Information

Below are hints in C language. These are just hints and not ready to use routines.

```
/*variables and defines*/

#define READ_DATA_COMMAND   100          /*read data command code*/

#define MAX_REGISTERS 128
#define MAXDATA             512          /*maximum number of bytes that labZY Tool will
                                         accept with a single write data command*/
#define S_AUTOINC    (1<<22)             /*auto-increment bit*/
#define SPECTRUM_BASE_ADDRESS      0x0000 /*base address of the spectrum data*/
#define REG_BASE_ADDRESS           0x8000 /*base address of the registers*/

#define READ_DATA_BYTE_OFFSET 24         /*FPGA data byte offset in the response from the
                                         labZY Tool*/
```

```
#define READ_MICRO_BYTE_OFFSET 8         /*MICRO data byte offset in the response from
                                          the labZY Tool*/
#define READ_DATA_WORD_OFFSET 12         /*FPGA data word offset in the response from the
                                          labZY Tool*/
#define READ_MICRO_WORD_OFFSET 4         /*MICRO data word offset in the response from
                                          the labZY Tool*/
#define READ_DATA_LONG_OFFSET 6          /*FPGA data long offset in the response from the
                                          labZY Tool*/
#define READ_MICRO_LONG_OFFSET 2         /*MICRO data long offset in the response from
                                          the labZY Tool*/

#define MAX_BYTES_RECEIVE 16500          /*maximum number of FPGA bytes to receive from
                                          labZY Tool in a single transmission*/

       union UNION_RCV_BUFFFER{
       unsigned char uc[MAX_BYTES_RECEIVE];
       unsigned short us[MAX_BYTES_RECEIVE/2];
       unsigned long ul[MAX_BYTES_RECEIVE/4];
       char          sc[MAX_BYTES_RECEIVE];
       short         ss[MAX_BYTES_RECEIVE/2];
       long          sl[MAX_BYTES_RECEIVE/4];
       } rbuf; /*host receive data buffer*/

       #define MAX_BYTES_SEND 532
       union UNION_SND_BUFFFER{
       unsigned char uc[MAX_BYTES_SEND];
       unsigned short us[MAX_BYTES_SEND/2];
       unsigned long ul[MAX_BYTES_SEND/4];
       char          sc[MAX_BYTES_SEND];
       short         ss[MAX_BYTES_SEND/2];
       long          sl[MAX_BYTES_SEND/4];
       } sbuf; /*host send data buffer*/

       #define MAX_BYTES_REGISTER 2*MAX_REGISTERS
       union UNION_REGISTERS{
       unsigned char uc[MAX_BYTES_REGISTER];
       unsigned short       us[MAX_BYTES_REGISTER/2];
       unsigned long ul[MAX_BYTES_REGISTER/4];
       char   sc[MAX_BYTES_REGISTER];
       short  ss[MAX_BYTES_REGISTER/2];
       long   sl[MAX_BYTES_REGISTER/4];
       } reg_mca; /*register storage*/


/********* READ COMMAND *********/
unsigned short readnanoMCA(unsigned long ul_Address, unsigned short us_DataBytesToRead,
              BOOL b_Autoinc){

              sbuf.us[0]=READ_DATA_COMMAND;      /*read data command code*/
              sbuf.us[1]=11;                     /*total number of bytes in the command*/
              if (b_autoinc) sbuf.ul[1]=ul_Address | AUTOINC; /*read this address
                                                               and auto-increment it*/
```

```
            else   sbuf.ul[1]=ul_Address & ~AUTOINC;  /*read this address
                                                       multiple times*/
            sbuf.us[4]=     us_DataBytesToRead;       /*number of bytes to be read*/

        /********** check sum calculation **********/
            sbuf.uc[10]=sbuf.uc[0];
            for (i=1; i< 10 ; i++){
                  sbuf.uc[10] += sbuf.uc[i];
            }
            sbuf.uc[10] = ~sbuf.uc[10]+2;

        /********** send data via serial port **********/
        /* use routine that is supported by OS, compiler, library etc.*/
        /* For example:  SerialPort.Write( &sbuf.c[0], sbuf.s[1],2000  );
        arguments(pointer to the first byte to be sent to serial port, number of bytes to
        be sent, time-out in mseconds)*/
        /**********************************************/

        return Something; /*Return something that is useful*/
        }


/****************** example using READ COMMAND ****************************/
unsigned short us_BytesToRead;
unsigned long ul_Address;
unsigned long ul_StartRegister;
unsigned short us_NumberRegisterstoRead;
us_NumberRegisterstoRead=127;
ul_StartRegister =1;
        ul_Address=REG_BASE_ADDRESS+ul_StartRegister;
        us_BytesToRead =readnanoMCA(ul_Address, us_NumberRegisterstoRead*2, S_AUTOINC);
/*read all nanoMCA registers, excluding register 0, that is read 127 16-bit registers =
254 bytes*/
        us_BytesToRead =readnanoMCA(SPECTRUM_BASE_ADDRESS, 8192, S_AUTOINC); /*read the
content of spectrum spectrum channels from channel 0 to channel 1023 inclusive (each
channel data has four bytes - unsigned long)*/
        us_BytesToRead =readnanoMCA(SPECTRUM_BASE_ADDRESS+10000, 8000, S_AUTOINC); /*read
the content of spectrum channels from channel 2500 to channel 4049 inclusive*/

/*************************************************************************/

/************************************************************************
Start a separate tread to read serial port data or use messages to read the serial port
data.
Place sequentially all bytes in the receive buffer buffer rbuf, with the first received
byte placed in rbuf.uc[0]
Read serial port data until the number of the received bytes matches us_BytesToRead
once the data is received:

1) Find the check sum
example:*/
unsigned char ucChkSum=0;
        for (int i=0; i< READ_FPGA_BYTE_OFFSET+(int)us_DataBytesToRead ; i++){
                    ucChkSum += rbuf.uc[i];
              }
```

```
       ucChkSum=~ucChkSum+2;

       if(ucChkSum != rbuf.uc[READ_FPGA_BYTE_OFFSET +
(int)us_DataBytesToRead]){handleError();}
       else (extractData());

/*2)Extract data
Example - populate the register union with the data read from the nanoMCA using
us_NumberRegisterstoRead and
ul_StartRegister*/

              for (i=0; i < us_NumberRegisterstoRead; i++){

                     reg_mca.us[ul_StartRegister+i] = rbuf.us[READ_DATA_WORD_OFFSET+i];

              }
```

Revision History

Tracking of the revision history begins with Rev A1