Networking in 1 Day!

NBPKBU-MMSCR Quad UART RS-232/485 Blade

Programming Reference Guide

Revision 1.0
Feburary 1, 2010

Table of Contents

1. Introduction

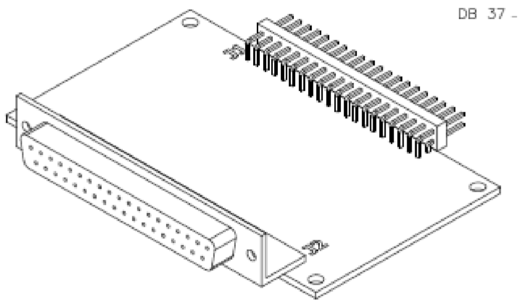
The NBPKBU-MMSCR is a personality blade board for the NBPK70EX-100CR. A Quad UART combined with four MAX3160E programmable RS-232/RS-485/422 multiprotocol transceivers provide four serial input/output channels that can be configured to any of the three protocols through software.

NOTE: Due to the design of the mult-protocol transceiver IC's, the RS-485/422 pinout differs slightly from the NetBurner NBPKBU-485CR blade board.

2. Installation

The NBPKBU-MMSCR is mounted inside the PK70 enclosure. It has two connectors: a DB37 (J2) that connects to external devices, and a dual-row, 40-pin right-angle header (J1) that connects the NBPKBU-MMSCR to the PK70 interface connector.

To install the NBPKBU-MMSCR, remove the PK70 cover, plug the 40-pin J1 header into the 40-pin socket on the PK70, and install the four 4-40 mounting screws. Finally, replace the PK70 cover and cover screws.

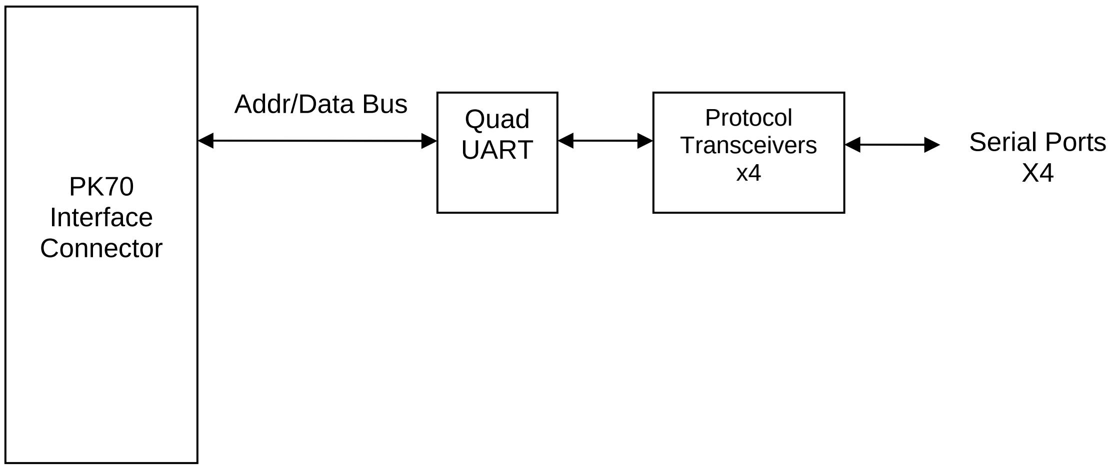


The software libraries are automatically installed with your PK70 development kit.

3. Hardware

The NBPKBU-MMSCR board interfaces to the PK70 through the 40-pin interface connector. This interface connector includes the address bus, data bus, QSPI, I^{2}C, clock, reset, chip selects, and interrupt signals. The blade board uses the parallel address/data bus to interface to an Exar XR16L784CV-F quad UART.

Block Diagram



4. Connector Pinouts

The NBPKBU-MMSCR has a DB37 female connector, with a pinout designed to follow the ribbon cable style DB9 male crimp type connectors. Please refer to section 10.7 of the NBPKBU-MMSCR User's Manual, located in:

\nburn\docs\platform\PK70\NBPKBU-485-UsersManual.pdf

for the quad UART cable signal configuration between the DB37 and four DB9 ports. For the pinout of the 40-pin J1 header that connects to the PK70 module, refer to the PK70 Hardware Manual (NBPK70.pdf), located in the same directory as the NBPKBU-MMSCR User's Manual.

5. Application Programming Interface

The following functions enable programming of the NBPKBU-MMSCR. The source code for this library are located in the \Nburn\PK70\include\NBPKQuadMMS.h and \Nburn\PK70\system\NBPKQuadMMS.cpp.

5.1 Serial Port Numbering Convention

The serial port numbering can be a bit confusing when looking at the hardware or schematics versus the software drivers. The ports are numbered from 1 to 4 when referencing hardware, including the quad UART cable. To maintain software conventions with other NetBurner serial drivers, the ports are numbered from 0 to 3 in the software drivers.

5.2 Baud Rate Calculation

When you specify a baud rate in the `PK70QuadMultiModeOpenSerial()` function, the baud rate for each port is calculated based on the quad UART crystal frequency as shown below:

```
divider = 14745600 / ( baud * 16 )
```

For example, the divider for a baud rate of 115,200 bits per second is eight. You can achieve any baud rate with a whole number divider value.

5.3 Open a Serial Port

Description:

Opens a serial port and returns a file descriptor if successful. Once a serial port is open you then need to configure it as RS-232, RS-485 or RS-422 with the appropriate library function.

Syntax:

```
int PK70QuadMultiModeOpenSerial( int portnum,
                                 unsigned int baudrate,
                                 int stop_bits,
                                 int data_bits,
                                 parity_mode parity );
```

Parameters:

Parameter	Type	Description
`portnum`	int	UART to open; valid values are 0-3.
`baudrate`	unsigned int	Baud rate in bits per second.
`stop_bits`	int	Number of stop bits; valid values are 1 and 2.
`data_bits`	int	Number of data bits; valid values are 5-8.
`parity`	parity_mode	Valid values are eParityNone, eParityOdd, and eParityEven.

Returns:

Value	Description
(fd > 0)	File descriptor associated with the opened serial port if successful.
-1	SERIAL_ERR_NOSUCH_PORT
-3	SERIAL_ERR_PORT_ALREADYOPEN
-4	SERIAL_ERR_PARAM_ERROR (returned if stop/data bit value or parity mode is invalid)

A simpler version of this function is also available. The only parameters required are the port number and baud rate. The stop bits, data bits, and parity are automatically set to 1, 8, and eParityNone, respectively.

```
int PK70QuadMultiModeSimpleOpenSerial( int portnum,
                                       unsigned int baudrate );
```

5.4 Close a Serial Port

Description:

This function closes a serial port that is currently open.

Syntax:

```
int PK70QuadMultiModeSerialClose( int portnum );
```

Parameters:

Parameter	Type	Description
`portnum`	int	UART to close; valid values are 0-3.

Returns:

Value	Description
0	UART successfully closed.
-1	SERIAL_ERR_NOSUCH_PORT
-2	SERIAL_ERR_PORT_NOTOPEN

5.5 Enable RS-485 or RS-422 Mode

Description:

Calling either the half-duplex or full-duplex function will configure the serial port in RS-485 half-duplex or full-duplex mode, respectively. One of these functions must be called after opening the serial port to enable RS-485 mode.

Syntax:

```
void PK70QuadMultiModeSerialSetFullDuplex485( int port );
void PK70QuadMultiModeSerialSetHalfDuplex485( int port,
                                               BOOL bEcho = FALSE);
```

Parameters:

Parameter	Type	Description
`port`	int	UART to enable half or full-duplex on; valid values are 0-3.
`bEcho`	BOOL	Setting to “FALSE” disables echo; setting to “TRUE” enables echo. This parameter only applies to the half-duplex enabling function. If this parameter is not explicitly provided in the function call, echo is disabled by default.

Returns:

Nothing to return.

5.6 Enable RS-232 Mode

Description:

Configure the specified serial port in RS-232 mode.

Syntax:

```
void PK70QuadMultiModeSerialSetRS232( int port );
```

Parameters:

Parameter	Type	Description
`port`	int	UART to enable RS-232; valid values are 0-3.

Returns:

Nothing to return.

5.7 Software Flow Control

Description:

These functions enable and disable the sending (Rx flow control) or acknowledgement (Tx flow control) of XON/XOFF flow control characters.

Syntax:

```
void PK70QuadMultiModeSerialEnableTxFlow( int port, int enab );
void PK70QuadMultiModeSerialEnableRxFlow( int port, int enab );
```

Parameters:

Parameter	Type	Description
`port`	int	UART to enable/disable flow control on; valid values are 0-3.
`enab`	int	‘0’ disables flow control; any non-zero value will enable it.

Returns:

Nothing to return.

5.8 RS-232 Hardware Flow Control

Description:

These functions enable and disable the sending of request-to-send (RTS) signals to regulate incoming data (Rx flow control) or receiving of clear-to-send (CTS) signals to throttle transmission (Tx flow control).

Syntax:

```
void PK70QuadSerialEnableHwTxFlow( int port, int enab );
void PK70QuadSerialEnableHwRxFlow( int port, int enab );
```

Parameters:

Parameter	**Type**	**Description**
`port`	int	UART to enable/disable flow control on; valid values are 0-3.
`enab`	int	‘0’ disables flow control; any non-zero value will enable it.

Returns:

Nothing to return.

5.9 Send Break

Description:

Sets a break in the UART transmission for a given period of time.

Syntax:

```
void PK70QuadMultiModeSerialSendBreak( int port,
                                       DWORD time );
```

Parameters:

Parameter	**Type**	**Description**
`port`	int	UART to set the break transmission on; valid values are 0-3.
`time`	DWORD	The length of time that the transmission break will occur, in ticks per second (20 ticks = 1 second by default).

Returns:

Nothing to return.

5.10 Read State of Carrier Detect

Description:

Gets the current state of the carrier detect (CD) pin from a specified UART.

Syntax:

```
BOOL PK70Quad MultiModeSerialGetCD( int port );
```

Parameters:

Parameter	Type	Description
port	int	UART associated with the CD pin to read; valid values are 0-3.

Returns:

Value	Description
TRUE	Carrier Detect signal is asserted.
FALSE	Carrier Detect signal is either negated or the given port is invalid.

5.11 Read State of Ring Indicator

Description:

Gets the current state of the ring indicator (RI) pin from a specified UART.

Syntax:

```
BOOL PK70QuadMultiModeSerialGetRI( int port );
```

Parameters:

Parameter	Type	Description
`port`	int	UART associated with the RI pin to read; valid values are 0-3.

Returns:

Value	Description
TRUE	Ring Indicator signal is asserted.
FALSE	Ring Indicator signal is either negated or the given port is invalid.

5.12 Read State of Data Set Ready

Description:

Gets the current state of the data set ready (DSR) pin from a specified UART.

Syntax:

```
BOOL PK70QuadMultiModeSerialGetDSR( int port );
```

Parameters:

Parameter	Type	Description
`port`	int	UART associated with the DSR pin to read; valid values are 0-3.

Returns:

Value	Description
TRUE	Data Set Ready signal is asserted.
FALSE	Data Set Ready signal is either negated or the given port is invalid.

5.13 Read State of Clear to Send

Description:

Gets the current state of the clear-to-send (CTS) pin from a specified UART.

Syntax:

```
BOOL PK70QuadMultiModeSerialGetCTS( int port );
```

Parameters:

Parameter	Type	Description
`port`	int	UART associated with the CTS pin to read; valid values are 0-3.

Returns:

Value	Description
TRUE	Clear-to-Send signal is asserted.
FALSE	Clear-to-Send signal is either negated or the given port is invalid.

5.14 Write State of Data Terminal Ready

Description:

Manually sets or clears the data terminal ready (DTR) pin for a specified UART.

Syntax:

```
void PK70QuadMultiModeSerialSetDTR( int port, BOOL val );
```

Parameters:

Parameter	Type	Description
`port`	int	UART associated with the DTR pin to set/clear; valid values are 0-3.
`val`	BOOL	“TRUE” asserts the signal; “FALSE” negates it.

Returns:

Nothing to return.

5.15 Write State of Request to Send

Description:

Manually sets or clears the request-to-send (RTS) pin for a specified UART.

Syntax:

```
void PK70QuadMultiModeSerialSetRTS( int port, BOOL val );
```

Parameters:

Parameter	Type	Description
`port`	int	UART associated with the RTS pin to set/clear; valid values are 0-3.
`val`	BOOL	“TRUE” asserts the signal; “FALSE” negates it.

Returns:

Nothing to return.