



DataBridge Serial Multiplexer (SDR-MUX)
User's Manual

Revision 1.0 • 1 May 2000

1

Getting Started

1.1 Getting Started

To get started you will need the DataBridge SDR, SDR-ACDC power supply, serial cables, the Omega D1132 modules, the RS-422 meteorological station, a computer with a free serial port, and terminal emulation software. Details of the following procedure can found in the DataBridge SDR User's Manual and in Sections 1.2-1.4.

1. Connect the serial cable to a terminal or computer running terminal software, connect the power and serial cables to the SDR, and verify that you see the SDR's main menu. See Section 1.4 of the DataBridge SDR User's Manual if you have any trouble.
2. Connect the meteorological station and Omega D1132 modules to the RS-422/485 ports. Be sure to maintain the proper polarity of the Data+ and Data- signals. Figure 1.1.1 shows how these devices should be connected to the SDR rear panel. Power supplies and the RS-232 serial cable have been omitted for clarity.
3. To verify data is being received, enter serial passthrough mode from the SDR main menu to talk to the SDR-MUX. From the SDR-MUX main menu, type **T** to test the script interactively, and begin monitoring the data flow. At this point, the incoming data should be displayed on the screen. The data LED on the SDR's front panel should also flash when data is being received. If the data looks correct, everything is connected

properly for recording data. To exit this mode type **Q** to stop the script and return to the SDR-MUX main menu. Typing the serial passthrough escape sequence, **+++**, will return you to the SDR main menu.

4. To record data (i.e. enter record mode), type **R** from the SDR main menu or press the record button on the SDR's front panel. The SDR is preconfigured to send an **R** to the SDR-MUX when entering record mode, which will automatically start the SDR-MUX script. When recording has stopped (i.e. when entering stop mode), the SDR will send a **Q**, which will halt script execution.

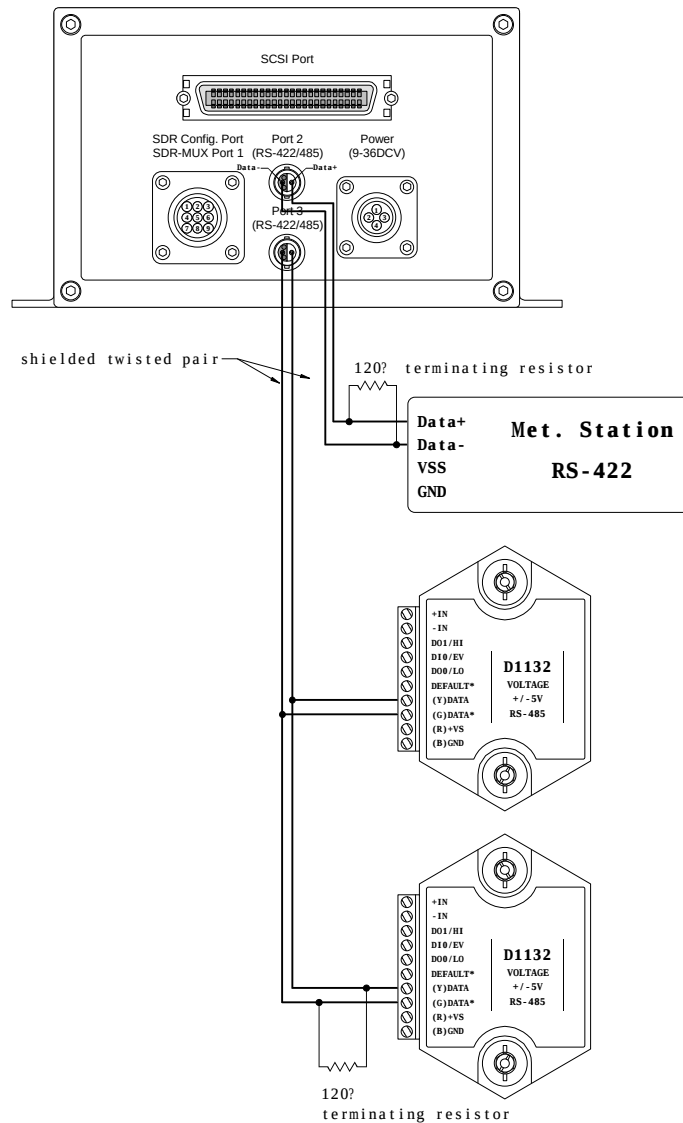


Figure 1.1.1. Connection of RS-422 Meteorological Station and two multi-drop RS-485 OmegaBus modules to DataBridge SDR.

1.2 Description

The DataBridge SDR Serial Multiplexer (SDR-MUX) is an add-on board for the base DataBridge SDR system that is designed to multiplex two input serial ports onto the SDR's single data port. The SDR-MUX is housed in the SDR enclosure and receives its power from the SDR circuit board. The SDR-MUX is equipped with four serial ports: one RS-232 port (Port 0) that is connected internally to the standard data port on the SDR circuit board, one auxiliary RS-232 port (Port 1) that is routed through the rear panel serial port connector, and two isolated half-duplex RS-422/485 ports (Ports 2 and 3) for incoming data. The auxiliary port can be used by your script for monitoring and data routing functions [see Section 2].

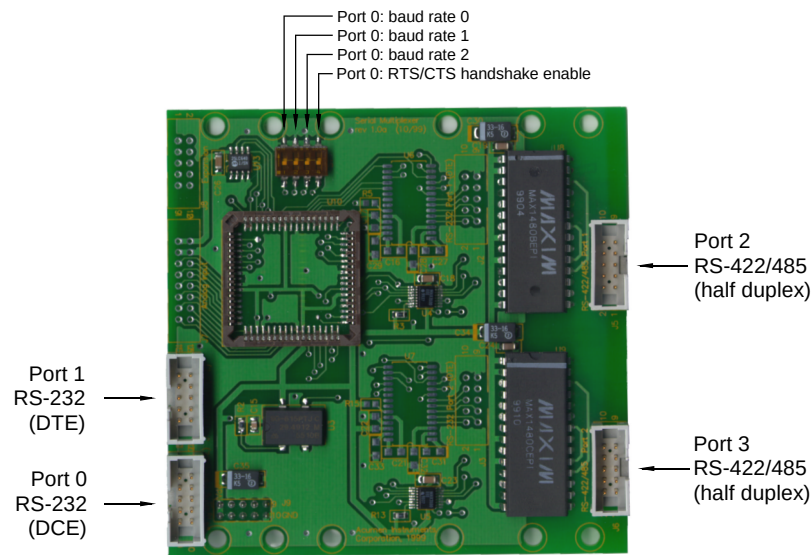


Figure 1.2.1. The serial multiplexer (SER-MUX) circuit board.

The SDR-MUX's behavior is controlled by a script that is loaded and executed by the SDR-MUX. This allows the SDR-MUX to adapt to a number of data recording configurations. For most applications, the SDR-MUX is shipped pre-configured. See the following sections for details on the configuration and editing of scripts, and for details about serial port configuration.

1.3 Operation

The SDR-MUX menus are accessed by entering serial passthrough mode on the SDR. Type **8** from the SDR's main menu to enter serial passthrough mode (Figure 1.3.1). You can return to the SDR main menu at any time by typing **+++** or by pressing the Stop button on the front panel of the SDR.

```
Acumen Instruments Corporation      12/14/1999 02:04:54
DataBridge SDR firmware rev. 5.2   Current filename is: BRIDGE.DAT

1 Set time
2 Set date
3 Toggle ANSI mode (ANSI is ON)
4 Toggle messages (messages are ON)

5 Edit messages...
6 SCSI drive functions...
7 File system functions...

8 Enter serial passthrough mode
9 Set baud rate for attached device

P Enter PLAY mode
R Enter RECORD mode

Enter choice (1,2,3,4,5,6,7,8,9,P,R) 8
Entering passthrough mode. Type +++ or press the STOP button to exit.
```

Figure 1.3.1. Entering serial passthrough mode from the DataBridge SDR main menu.

To display the SDR-MUX main menu (Figure 1.3.2) you need to press the **<spacebar>**. The SDR-MUX main menu allows you to configure the serial ports, view and edit the current script, test the script interactively, and turn ANSI graphics mode on and off.

```
Acumen Instruments Corporation
DataBridge SDR serial multiplexer rev. 1.0

1 Configure port: Aux Port
2 Configure port: MetStation Port
3 Configure port: OmegaBus Port

S View/edit script...
T Test script interactively
A Change ANSI graphics mode (ANSI is on)

Enter choice (1,2,3,S,T,A)
```

Figure 1.3.2. SDR-MUX main menu via the SDR's serial passthrough mode.

1.3.1 Configuring the serial ports

Serial Port 0 is configured using the DIP switches located on the SDR-MUX circuit board. See Figure 1.3.3 for settings.

Label	baud rate 0	baud rate 1	baud rate 2	RTS/CTS handshake enable
Switch Designator	SW1	SW2	SW3	SW4
RTS/CTS hardware handshaking enable	-	-	-	ON
230400 bps	OFF	OFF	OFF	-
115200 bps	ON	OFF	OFF	-
57600 bps	OFF	ON	OFF	-
38400 bps	ON	ON	OFF	-
19200 bps	OFF	OFF	ON	-
9600 bps	ON	OFF	ON	-
4800 bps	OFF	ON	ON	-
2400 bps	ON	ON	ON	-

Figure 1.3.3. Serial port 0 baud rate settings.

To configure the auxiliary RS-232 DTE port, named Aux Port in this example, type **1**. The SDR-MUX presents you with a menu that allows you to set the port speed. On this port, the data format is always 8 data bits, no parity, and one stop bit (8N1). To return to the SDR-MUX main menu type **<ESC>** or **<ENTER>**.

```

Acumen Instruments Corporation
DataBridge SDR serial multiplexer rev. 1.0
- Aux Port
1 Confi|
2 Confi| Port Speed (bps)
3 Confi| A: 230400 E: 19200
      | B: 115200 F: 9600
S View/| C: 57600 G: 4800
T Test | D: 38400 H: 2400
A Chang|
      | Current settings: 115200 8N1
Enter cho|

```

Figure 1.3.4. Changing the port speed for the auxiliary RS-232 port.

The port labeled MetStation Port can be configured by typing **2** from the SDR-MUX main menu and the OmegaBus Port can be configured by typing **3** from the SDR-MUX main menu. Both of these menu items allow you to

configure port speed, number of data bits, type of parity, and number of stop bits. Configure these ports to communicate with the equipment will be recording data from. See your equipment's documentation to determine a compatible port speed and data format. Typing **<ESC>** or **<ENTER>** returns you to the SDR-MUX main menu.

```

Acumen Instruments Corporation
DataBridge SDR serial multiplexer rev. 1.0
  _ MetStation Port
1 Conf|
2 Conf| Port Speed (bps)      Parity
3 Conf| A: 230400    J: 7200    0: None
      B: 115200    K: 4800    1: Even
S View| C: 76800     L: 3600    2: Odd
T Test| D: 57600    M: 2400    3: Mark
A Chang| E: 38400    N: 1800    4: Space
      F: 28800    O: 1200
Enter cho| G: 19200    P: 900     Stop Bits
      H: 14400    Q: 600     5: 1
      I: 9600     R: 300     6: 2
                        Data Bits
                        7: 7
                        8: 8

      Current settings: 9600 8N1

```

Figure 1.3.5. Changing the port settings for RS-422/485 port #1.

1.3.2 Viewing and editing the current script

To view and/or edit the script, type **S** from the SDR-MUX main menu.

Figure 1.3.6 shows the Edit Script menu. When editing a script, the script name appears at the top of the screen followed by a text presentation of the script. If a script is too long to display on one screen, typing **A** and **Z** will scroll the screen up and down, respectively.

```

Script name: Army_LS.mux

If data from MetStation Port is "\n"
then increment line counter
and send "L" via DataBridge Port
and send line counter via DataBridge Port
and send " " via DataBridge Port
and pause for 5 milliseconds
and send MetStation Port's buffer contents via DataBridge Port
and set OmegaBus_Addr to 1
and send "#0RD\r" via OmegaBus Port
and pause for 50 milliseconds
and send "L" via DataBridge Port
and send line counter via DataBridge Port
and send " " via DataBridge Port
and send OmegaBus Port's buffer contents via DataBridge Port
and send "\n" via DataBridge Port
If OmegaBus_Addr=1
then set OmegaBus_Addr to 0
and send "#1RD\r" via OmegaBus Port
and pause for 50 milliseconds
(script continued below... use Z to view additional lines)
U Upload a script      A Scroll up          Z Scroll down
P Change port names   V Change variable names Q Return to main menu

```

Figure 1.3.6. The View/edit script... menu.

1.3.3 Changing port and variable names

By typing **P** from the Edit Script menu, you can modify port names. Type **Q** to return to the Edit Script menu. Likewise, typing **V** from the edit script menu allows you to modify any variable names used by the script. Once again, **Q** returns you to the edit script menu.

```
Script name: Army_LS.mux

If data _ Port Names _____
then i|
and s| Port 0 is called: DataBridge Port
and s| Port 1 is called: Aux Port
and s| Port 2 is called: MetStation Port
and p| Port 3 is called: OmegaBus Port
and s|
and s| Enter port number to modify, Q to quit
and s|
and p| _____ Port
and send "L" via DataBridge Port
and send line counter via DataBridge Port
and send " " via DataBridge Port
and send OmegaBus Port's buffer contents via DataBridge Port
and send "\n" via DataBridge Port
If OmegaBus_Addr=1
then set OmegaBus_Addr to 0
and send "#1RD\r" via OmegaBus Port
and pause for 50 milliseconds
(script continued below... use Z to view additional lines)
U Upload a script      A Scroll up      Z Scroll down
P Change port names    V Change variable names  Q Return to main menu
```

Figure 1.3.7. Modifying port names.

```
Script name: Army_LS.mux

If data _ Variable Names _____
then i|
and s| Variable 0 is called: OmegaBus_Addr
and s| Variable 1 is called: time counter
and s| Variable 2 is called: line counter
and p| Variable 3 is called: k
and s|
and s| Enter variable to modify, Q to quit
and s|
and p| _____ Port
and send "L" via DataBridge Port
and send line counter via DataBridge Port
and send " " via DataBridge Port
and send OmegaBus Port's buffer contents via DataBridge Port
and send "\n" via DataBridge Port
If OmegaBus_Addr=1
then set OmegaBus_Addr to 0
and send "#1RD\r" via OmegaBus Port
and pause for 50 milliseconds
(script continued below... use Z to view additional lines)
U Upload a script      A Scroll up      Z Scroll down
P Change port names    V Change variable names  Q Return to main menu
```

Figure 1.3.8. Modifying variable names.

1.3.4 Uploading a new script

To upload a new script, type **U** from the edit script menu. Since this operation will overwrite the current script, the SDR-MUX prompts for confirmation of the upload (Figure 1.3.9). If you are sure you want to upload a new script, type **Y** and ASCII upload your new script using your terminal emulation software. If you cancel the upload from your terminal software you may need to power cycle the SDR-MUX so you can upload a script again. You can escape before you begin the upload by typing any

character other than **Y** when prompted. Your terminal software should have the line pacing or line delay set to 100 milliseconds or longer.

```
Script name: Army_LS.mux

If data from MetStation Port is "\n"
  then increment line counter
  and send "L" via DataBridge Port
  and send line counter via DataBridge Port
  and send " " via DataBridge Port
  and pause for 5 milliseconds
  and send MetStation Port's buffer contents via DataBridge Port
  and set OmegaBus_Addr to 1
  and send "#0RD\r" via OmegaBus Port
  and pause for 50 milliseconds
  and send "L" via DataBridge Port
  and send line counter via DataBridge Port
  and send " " via DataBridge Port
  and send OmegaBus Port's buffer contents via DataBridge Port
  and send "\n" via DataBridge Port
If OmegaBus_Addr=1
  then set OmegaBus_Addr to 0
  and send "#1RD\r" via OmegaBus Port
  and pause for 50 milliseconds
(script continued below... use Z to view additional lines)
Ready for ASCII upload. If uploading is cancelled, reset the SDR-MUX.
Press Y to continue or any other key to cancel upload...
```

Figure 1.3.9. Beginning an ASCII upload of a new script.

1.3.5 Testing the current script

Typing **T** from the SDR-MUX main menu allows you to run the current script and display the SDR-MUX output in real time. When entering this mode the SDR-MUX will look for a stop action and, if it finds one, will remind you what it is so you can stop the script and exit interactive testing mode. This is important because a script will run indefinitely without a stop action (i.e. the SDR-MUX would have to be power cycled to stop the script). If, by some chance, a script was uploaded that did not have a stop action, the SDR-MUX will warn you and not run the script. Press any key to start the script.

```
The current script's primary stop instruction is:

If data from DataBridge Port is "Q"
  then Stop script and return to menu mode

Press any key to run the script interactively...
```

Figure 1.3.10. Testing a script interactively. Notice that the SDR-MUX reminds you what the primary stop instruction is.

2

Scripting Reference

2.1 Introduction

Scripting allows you to modify the SDR-MUX behavior to suit a specific application. This section will attempt to give the reader some understanding of how to write, “compile”, and upload scripts for the SDR-MUX. Currently there are no software tools to assist in writing and compiling scripts, as scripts were originally intended for in-house use only. We have decided to allow access to scripting as a service to our customers.

2.2 Scripting Examples

Figure 2.2.1 shows the full Army_LS.mux script as interpreted and displayed by the SDR-MUX:

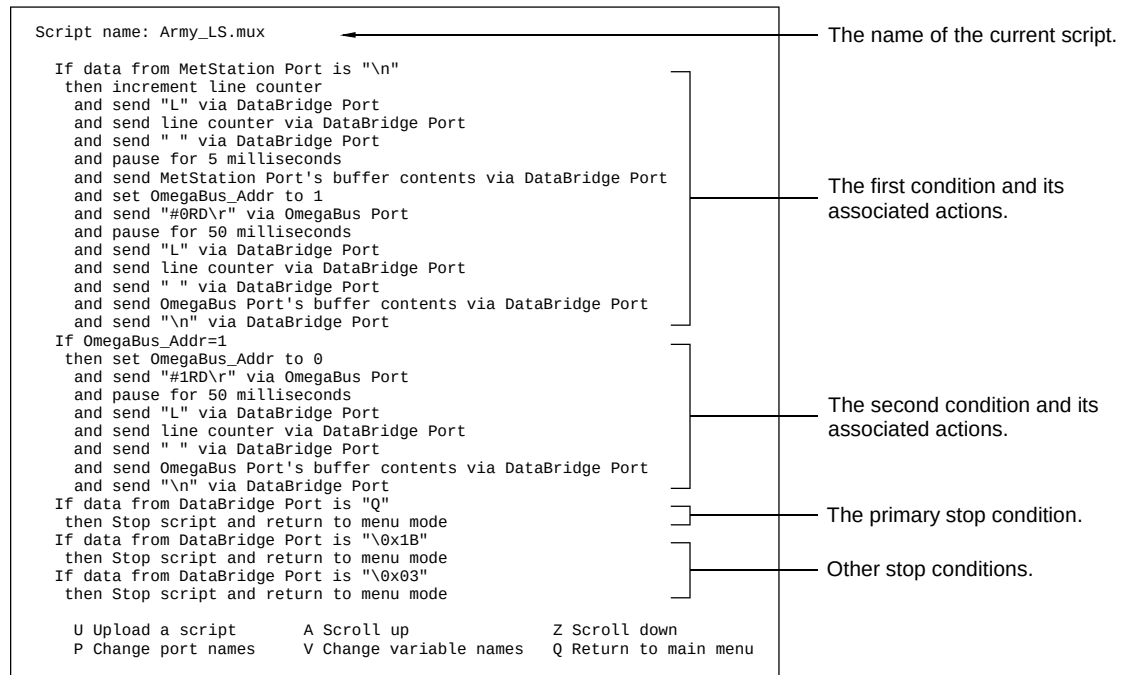


Figure 2.2.1. SDR-MUX display of the entire Army_LS.mux script.

2.3 Condition Opcodes

if data from port 0 is <i>string</i>															
(where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
128	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

if data from port 1 is <i>string</i>															
(where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
129	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

if data from port 2 is <i>string</i>															
(where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
130	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

if data from port 3 is <i>string</i>															
(where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
131	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

if port 0's buffer contains more than <i>n</i> bytes															
(where <i>n</i> is a 16-bit little-endian integer)															
Opcode	Operand														
144	<i>n0</i>	<i>n1</i>	-	-	-	-	-	-	-	-	-	-	-	-	-

if port 1's buffer contains more than <i>n</i> bytes															
(where <i>n</i> is a 16-bit little-endian integer)															
Opcode	Operand														
145	<i>n0</i>	<i>n1</i>	-	-	-	-	-	-	-	-	-	-	-	-	-

if port 2's buffer contains more than <i>n</i> bytes															
(where <i>n</i> is a 16-bit little-endian integer)															
Opcode	Operand														
146	<i>n0</i>	<i>n1</i>	-	-	-	-	-	-	-	-	-	-	-	-	-

if port 3's buffer contains more than n bytes (where n is a 16-bit little-endian integer)															
Opcode	Operand														
147	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

if variable0 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
160	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

if variable1 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
161	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

if variable2 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
162	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

if variable3 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
163	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

if variable0 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
168	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

if variable1 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
169	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

if variable2 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
170	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

if variable3 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
171	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

if hardware handshake line on port 0 is asserted/deasserted															
Opcode	Operand														
*	val	-	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

deasserted				asserted			
RI	DCD	DSR	CTS	RI	DCD	DSR	CTS
b7	b6	b5	b4	b3	b2	b1	b0

where val =

if hardware handshake line on port 1 is asserted/deasserted															
Opcode	Operand														
*	val	-	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

deasserted				asserted			
RI	DCD	DSR	CTS	RI	DCD	DSR	CTS
b7	b6	b5	b4	b3	b2	b1	b0

where val =

if hardware handshake line on port 2 is asserted/deasserted															
Opcode	Operand														
*	val	-	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

deasserted				asserted			
RI	DCD	DSR	CTS	RI	DCD	DSR	CTS
b7	b6	b5	b4	b3	b2	b1	b0

where val =

if hardware handshake line on port 3 is asserted/deasserted															
Opcode	Operand														
*	val	-	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

deasserted				asserted			
RI	DCD	DSR	CTS	RI	DCD	DSR	CTS
b7	b6	b5	b4	b3	b2	b1	b0

where val =

2.4 Action Opcodes

send port n buffer contents via port 0 (where n the port number 0-3)															
Opcode	Operand														
16	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send port n buffer contents via port 1 (where n the port number 0-3)															
Opcode	Operand														
17	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send port n buffer contents via port 2 (where n the port number 0-3)															
Opcode	Operand														
18	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send port n buffer contents via port 3 (where n the port number 0-3)															
Opcode	Operand														
19	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send string via port 0 (where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
32	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

send string via port 1 (where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
33	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

send string via port 2 (where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
34	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

send string via port 3 (where <i>string</i> is a null-terminated ASCII string no longer than 14-bytes)															
Opcode	Operand														
35	<i>ch1</i>	<i>ch2</i>	<i>ch3</i>	<i>ch4</i>	<i>ch5</i>	<i>ch6</i>	<i>ch7</i>	<i>ch8</i>	<i>ch9</i>	<i>ch10</i>	<i>ch11</i>	<i>ch12</i>	<i>ch13</i>	<i>ch14</i>	0

set variable0 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
48	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

set variable1 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
49	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

set variable2 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
50	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

set variable3 = n (where n is a 32-bit little-endian integer)															
Opcode	Operand														
51	$n0$	$n1$	$n2$	$n3$	-	-	-	-	-	-	-	-	-	-	-

set variable0 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
56	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

set variable1 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
57	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

set variable2 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
58	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

set variable3 = n (where n is a 16-bit little-endian integer)															
Opcode	Operand														
59	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

add n to variable0															
(where n is a 32-bit little-endian integer)															
Opcode	Operand														
64	$n0$	$n1$	$n3$	$n4$	-	-	-	-	-	-	-	-	-	-	-

add n to variable1															
(where n is a 32-bit little-endian integer)															
Opcode	Operand														
65	$n0$	$n1$	$n3$	$n4$	-	-	-	-	-	-	-	-	-	-	-

add n to variable2															
(where n is a 32-bit little-endian integer)															
Opcode	Operand														
66	$n0$	$n1$	$n3$	$n4$	-	-	-	-	-	-	-	-	-	-	-

add n to variable3															
(where n is a 32-bit little-endian integer)															
Opcode	Operand														
67	$n0$	$n1$	$n3$	$n4$	-	-	-	-	-	-	-	-	-	-	-

add n to variable0															
(where n is a 16-bit little-endian integer)															
Opcode	Operand														
*	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

add n to variable1															
(where n is a 16-bit little-endian integer)															
Opcode	Operand														
*	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

add n to variable2															
(where n is a 16-bit little-endian integer)															
Opcode	Operand														
*	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

add n to variable3															
(where n is a 16-bit little-endian integer)															
Opcode	Operand														
*	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-

*proposed opcode, not implemented

send variable n via port 0															
(where n is the variable index 0-3)															
Opcode	Operand														
80	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send variable n via port 1															
(where n is the variable index 0-3)															
Opcode	Operand														
81	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send variable n via port 2															
(where n is the variable index 0-3)															
Opcode	Operand														
82	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

send variable n via port 3															
(where n is the variable index 0-3)															
Opcode	Operand														
83	n	-	-	-	-	-	-	-	-	-	-	-	-	-	-

delay n milliseconds															
(where n is a 16-bit little-endian integer)															
Opcode	Operand														
96	$n0$	$n1$	-	-	-	-	-	-	-	-	-	-	-	-	-