EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
ORGANIZATION EUROPÈENNE POUR LA RECHERCHE NUCLÈAIRE

CERN - AB DIVISION

AB-Note-2005-009

Software documentation, and specifications of the Analogue 48x8 Multiplexer

Levente Kovacs

Abstract

This document gives a description of the software operating in the Analogue 48x8 Multiplexer (AMUX). The AMUX switches analogue signals from its inputs to the outputs. It can handle low power signals, with bandwidth of 120MHz. The AMUX is controlled from an IP network, and uses the TCP, and a system specific upper layer protocol for communication with the controller peer. The operating protocol is detailed as well.

Geneva, Switzerland
23rd February 2005

Contents

1 Introduction **3**

2 Hardware **3**

3 Software **4**

3.1 Software running in the MCU of the AMUX 4

3.1.1 LCD handling library 4

3.1.2 path_control.c 5

3.1.3 mux_control.c 5

3.2 Networking library 6

3.2.1 The mux_address structure 6

3.2.2 open_mux() 6

3.2.3 setup() 6

3.2.4 bye() 7

3.2.5 Operations 7

3.3 Compilation 7

3.4 Graphic User Interface (GUI) 7

3.4.1 Compilation 8

3.4.2 Controls 8

3.4.3 Operations 8

3.4.4 Error monitoring 9

4 The protocol **9**

4.1 General aspects 9

4.2 Command overview 10

4.2.1 SETUP 10

4.2.2 CLEAR 10

4.2.3 QUERY 10

4.2.4 BYE 10

4.3 Operations 10

5 Measurements **11**

5.1 Frequency response 12

5.2 Typical delay 12

5.3 Typical difference of delays between input groups 12

5.4 Typical difference of delays between output groups 14

5.5 Time domain 14

5.6 Crosstalk 14

6 Specifications **16**

6.1 Interfaces 16

6.1.1 Input / output analogue interfaces 16

6.1.2 IEEE802.3 16

6.1.3 V.24 (RS232C) 16

6.2 General specifications 17

A Hardware errors **17**

B Hardware programming **18**

1 Introduction

The main principle of the Analogue 48x8 Multiplexer (AMUX) is to replace the relay arrays in the PS CODD calibration system. Those relays were switching low power high frequency signals. Since a huge number of relays were used to achieve the signal switching, the MTBF was too low. The AMUX has no mechanical device inside. The AMUX also helps local analogue signal observation and special purpose acquisition, and it is used in other processing equipment.

2 Hardware

The hardware was an existing design. The signal switching is achieved by semiconductor chips. Figure 1 shows the block diagram of the AMUX. It consists of six different blocks. The Input limiters, the multiplexers, output buffers, the control system, the LCD, and the power supply. These blocks are logical blocks, and can not be separated physically.

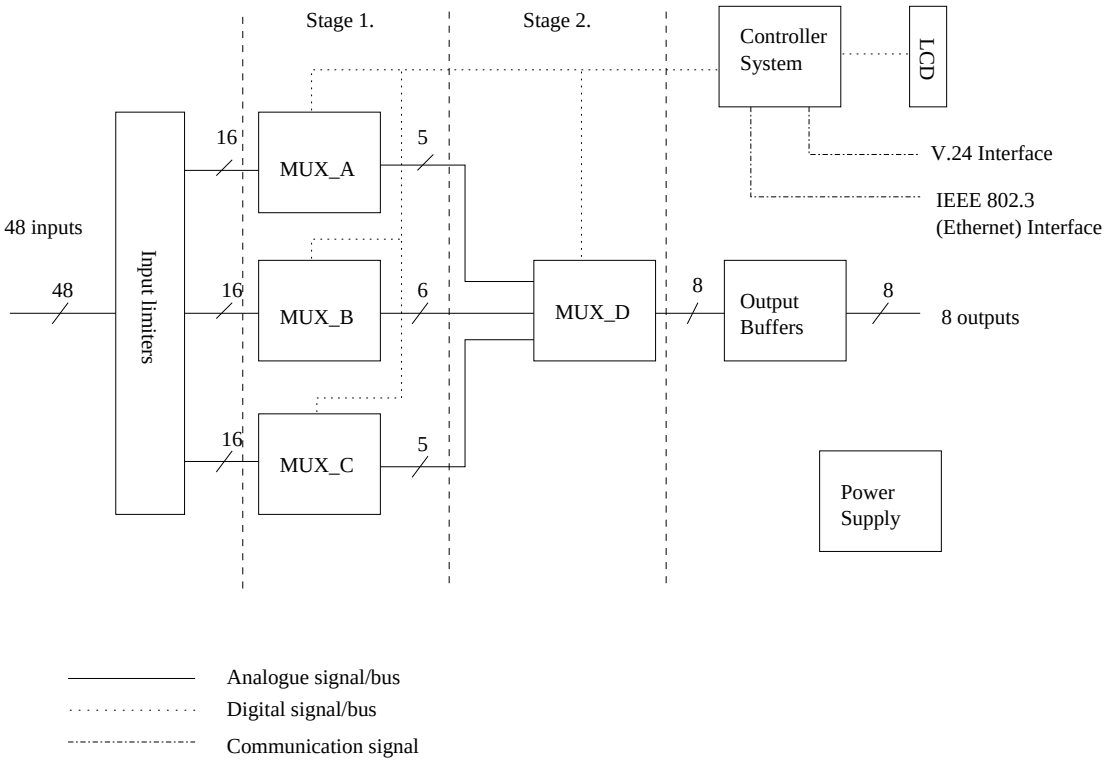

Figure 1: Block diagram of the AMUX

Each input is equipped with a limiter for protection against over-voltage. It does not let the input signal be greater or smaller than the positive and the negative power supply respectively.

The AMUX consists of two switch stages. At each stage the Analog Devices's AD8110 16x8 multiplexer integrated circuits are used to switch analogue signals. At the first stage, there are three multiplexer chips. Each handles 16 line out of the 48 inputs. As the second stage, a 4th chip collects the outputs of the first stage. Each output is buffered by an AD811 OPA.

The four multiplexer chips are controlled by a Rabbit RCM2100 MCU. It also provides network connection to ethernet networks, and controls the LCD.

The power supply consists of rectifiers, filter capacitors, and four linear regulator chips which are supplied by a transformer. The power supply provides ±5 and ±12 volts.

For more information on the hardware, see [1].

3 Software

3.1 Software running in the MCU of the AMUX

The software running on the MCU of the AMUX provides three functions:

- Controlling the multiplexer ICs
- Supervision of the internal lines between the two stages
- Protocol implementation for network communication

The **main.c** file implements the protocol, and some device initializations, and structure definitions. This file provides the main() function.

The main.c uses

- LCD_OK.c
- PATH_CONTROL.c

files to handle the LCD, and to manage internal analogue signal interconnections.

3.1.1 LCD handling library

LCD_OK.c implements two user functions, and low level hardware access functions.

1. int LCDprintf(char *str);
2. void LCDinit();

LCDprintf() prints a string to the LCD panel pointed to by ***str**. The string is not formatted, thus it must be formatted if needed. The other parts of the software use the sprintf() function to do so. See source for example. The pointed string must contain one *newline* (NL) character at most. The characters before the NL are displayed in the first row, while the others are displayed in the second row. If the pointed string contains more than one NL character, LCDprintf() returns 1, otherwise 0.

Note that the number of characters in a line are not checked. LCDprintf() overwrites the internal buffer of the LCD if too many characters are being printed. The LCD accepts two lines of 20 characters.

The LCDinit() function sets the LCD to the proper operation mode. Before calling the LCDprintf() function, the LCDinit() function must be called. It takes no arguments, and has void return value.

3.1.2 path_control.c

This file implements the main part of the software. This function is used to create internal paths by calling lower level functions (mux_control.c), and keeps track of internal connections. Paths can be created from any input to any output, or an output can be disconnected.

It tries to optimise the internal lines between the two stages. If one input signal must be distributed to two or more outputs, switching is done at the second stage, and only one internal line is occupied.

The only provided function is

```
unsigned char path_control(char from, char to, struct state_ *pp);
```

from argument is allowed to be an integer, from 0 to 47, and 64. **to** value is an integer from 0 to 7. ***pp** is a pointer to a struct _state structure. This structure is defined in main.c.

If this function is called with the **from** value is in the range of 0 and 47, it will try to create a path from the input to the addressed output.

If the **from** value is 64, it will set the addressed output to *not connected* (NC) state.

The path_control() function returns the following codes:

- 0 on success
- 1, if there is no internal line left for the connection
- 2, if the address was out of range

3.1.3 mux_control.c

This file implements a function to control the multiplexer integrated circuits physically.

```
int mux_ctrl(char mux, char from, char to);
```

mux is an integer from 0 to 3, where

- 0 is MUX_A (input 0 to 15)
- 1 is MUX_B (input 16 to 31)
- 2 is MUX_C (input 32 to 47)
- 3 is MUX_D (second stage).

from is an integer from 0 to 15. This is the input address of the multiplexer. **to** is an integer from 0 to 7. This is the output address of the multiplexer.

mux_ctrl() will always return 0.

3.2 Networking library

On the other side of the network, a UNIX machine controls the AMUX. The code is fully POSIX [3] compatible, so it can be compiled in any POSIX-like system. The development platform was Linux/i386. Apart from Linux/i386, the software is known to compile on Solaris/SPARC architecture, including the GUI.

This library has only one file, implementing the remote controlling procedures of the AMUX.

```
struct mux_address
{
   int sockfd;
   char address[60];
   int port;
};

int open_mux(struct mux_address *data);
int setup(struct mux_address *data, int from, int to);
int bye(struct mux_address *data);
```

Listing 1: Prototypes of the network library

3.2.1 The mux_address structure

The members of **mux_address** structure holds the address of the AMUX. The *address* member is a pointer to a string, which can be an IP address in dotted decimal notation, or a host name. The *port* member is an integer defines the destination IP layer port. Normally 2048.

The user must not touch the *sockfd* member.

3.2.2 open_mux()

This function is used to open a connection to the remote AMUX unit. open_mux() takes only one argument; a pointer to a **mux_address** type structure. The open_mux function returns 0 on success, otherwise -1.

3.2.3 setup()

This function is used to create or clear a path from one of the 48 inputs, to one of the outputs.

setup() takes 3 arguments:

1. a pointer to a **mux_address** type structure
2. **from** integer of range from 0 to 47 for input addressing or 64 for clearing
3. **to** integer of range from 0 to 7

The setup() function returns the following values:

- 0 on success
- -1, if there was no free route left to the addressed output
- -2, if one of the addresses (**from** or **to** values) is out of range
- -3, if a communication error occurred.

3.2.4 bye()

The bye() function closes the connection, which was created by the open_mux() call. bye() takes only one argument which is a pointer to the **mux_address** structure.

The bye() function returns the following values:

- 0 on success
- -1 otherwise

3.2.5 Operations

Usually the given sequence is performed:

1. a mux_address type structure must be initialized
2. the **address** and **port** members are filled with the corresponding values
3. the open_mux() function is called
4. setup() function is called in order to create and/or clear paths in the remote AMUX
5. the bye() function must be called to close the connection

If this library used as detailed above, the commands transmitted to the AMUX are satisfying the implemented protocol (See section 4).

and the structure.

3.3 Compilation

The **network.c** file is compiled against the standard C library. The **network.h** file must be included before using this code. For compiler option, see the Makefile coming with the distribution.

3.4 Graphic User Interface (GUI)

The GUI is for testing. If direct access is needed to the remote AMUX, a GTK+ client helps to control. Figure 2 shows the user interface.

3.4.1 Compilation

The software is compiled against the network.c file, and the GTK+-1.2.10 [4]. Both must be present on the system. Read and edit the Makefile to change the C compiler or its options settings[1]. Type "make" to build the program. "make install" places the executable to /usr/local/bin.

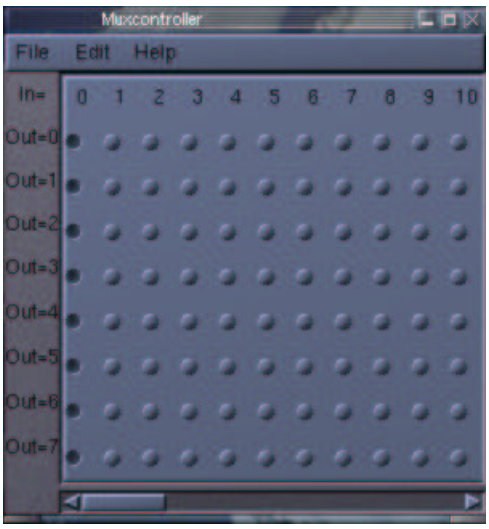

Figure 2: The GUI

3.4.2 Controls

The GUI offers the following controls:

- File menu
  Connection can be opened or closed by clicking the appropriate item. The program is terminated by clicking the Exit item.
- Edit menu
  Address and Port values can be set by selecting the Preferences item.
- Help menu
  The About... item shows information about the program, including the date of the compilation.

3.4.3 Operations

After the program starts, the appropriate Address and Port values must be set. Connection to the remote AMUX system must be created afterwards. Once the connection is established, paths can be created from inputs to outputs by clicking the appropriate matrix elements. Finally, the connection must be closed.

[1]The default setting for C compiler is gcc

3.4.4 Error monitoring

Error messages are generated using dialogue windows.

4 The protocol

The AMUX and the control system are communicating with the IP protocol. (Figure 3). The network library comes with the RABBIT processor (Dynamic C) implements the stack up till the 4th layer, namely TCP. A 5th layer protocol was designed to control the AMUX [2].

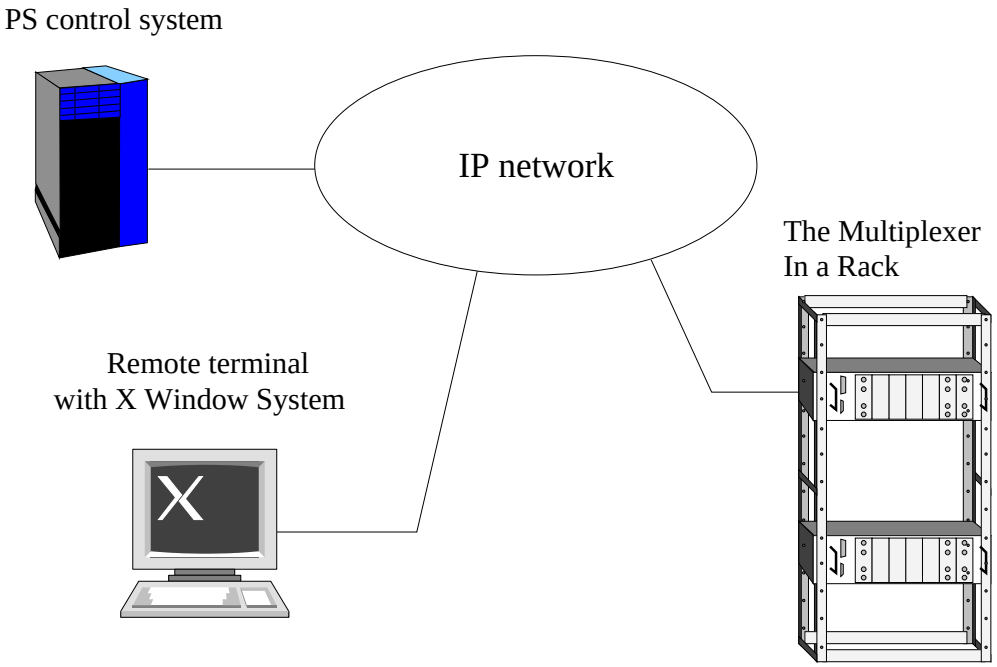

Figure 3: The multiplexer in CERN's environment

4.1 General aspects

The protocol is command oriented and clear text based to provide direct control over the AMUX. Commands may or may not have arguments. Each command is terminated by a "CR/LF" sequence. Each command sent is acknowledged by a response. Responses always started and ended with newline (NL) character. After the initial NL character, a + or - sign is transmitted, referring to positive or negative acknowledgement. After the sign, some literal information is transmitted.

The protocol may be used on a serial line. This is not yet implemented. Note that a raw serial line does not have any link layer capabilities. One should use HDLC, or PPP to implement this function. Bytes (messages) can be pushed to the line, but in that case, there is no chance to identify data losses, and the channel will be unreliable. This way is not recommended.

4.2 Command overview

4.2.1 SETUP

Syntax: **SETUP** *from to*

This command is used to make a connection from the input *from* to the output *to*, where *from*, and *to* are integer numbers referring the numbers of the input (0-47), and the output (0-7).

If the operation is successful, the response starts with "+" sign. Otherwise, "-". Apart from the signs, a literal response is sent. For example, a positive acknowledgement:

+OK Path created successfully. Internal path=0

Where *Internal path* is an integer representing the used line number between the two stages. (0 at the example above).

4.2.2 CLEAR

Syntax: **CLEAR** *to*

This command disconnects the output addressed by the argument. The *to* field identifies the target output. Value is an integer from 0 to 7.

4.2.3 QUERY

Syntax: **QUERY** [*to*]

This command will give a list of the internal connections. If *to* is given, it'll only return the addressed output.

4.2.4 BYE

Syntax: **BYE**

This command will close the connection.

4.3 Operations

Usually, the following process is performed.

1. A connection is created to the remote AMUX unit.
   This is done by connecting the TCP port of the Multiplexer (Normally:2048). You can issue a UNIX command like:

   telnet[2] <address> 2048

[2]The telnet program coming with Windows is NOT tested.

where <address> is the IP address of the multiplexer, or a host name. The second argument (2048) is the TCP port.

2. Commands can be sent such as

   - **SETUP**
   - **QUERY**
   - **CLEAR**

3. Finally the connection must closed by issuing the **BYE** command.

5 Measurements

The Hewlett Packard HP8753D Network analyzer was used for the measurements. Figure 4 shows the layout of the measurements. The input and output ports of the analyzer were connected to different outputs and inputs of the multiplexer.

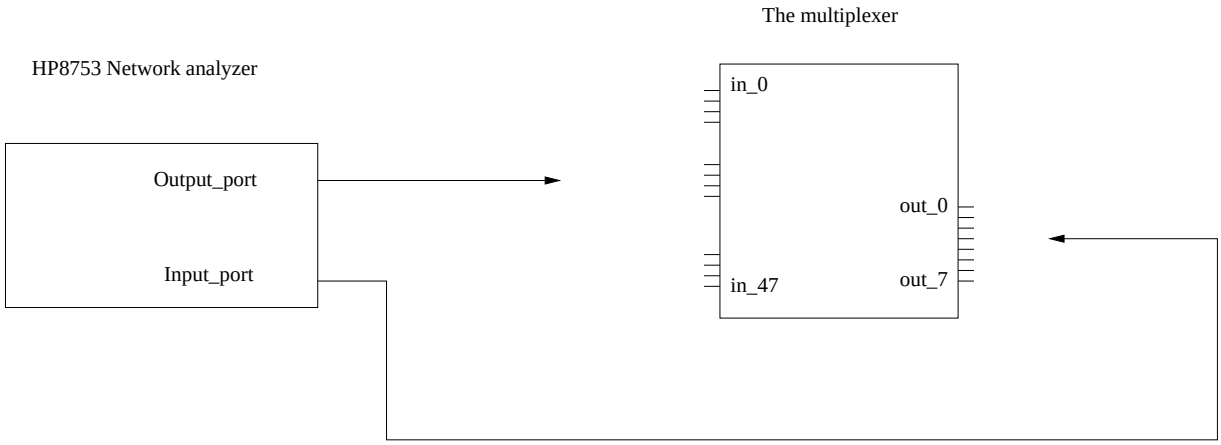

Figure 4: The measurement layout

The following measurements were recorded:

1. The frequency response;
2. the typical delay between one input and one output terminals;
3. the typical delay between two paths. Both inputs and outputs were in the same input group;
4. the difference of delays between two inputs belonging two different input multiplexer IC (two different input groups);
5. the difference of delays between two outputs belonging two different output groups;
6. time domain;
7. crosstalk between two adjacent paths.

5.1 Frequency response

The frequency response was measured by connecting creating one path, and connecting the analyzer to the input and the output of the path respectively. The analyzer was calibrated before the measurement. Figure 5. shows the plot of the typical frequency response of a created path.

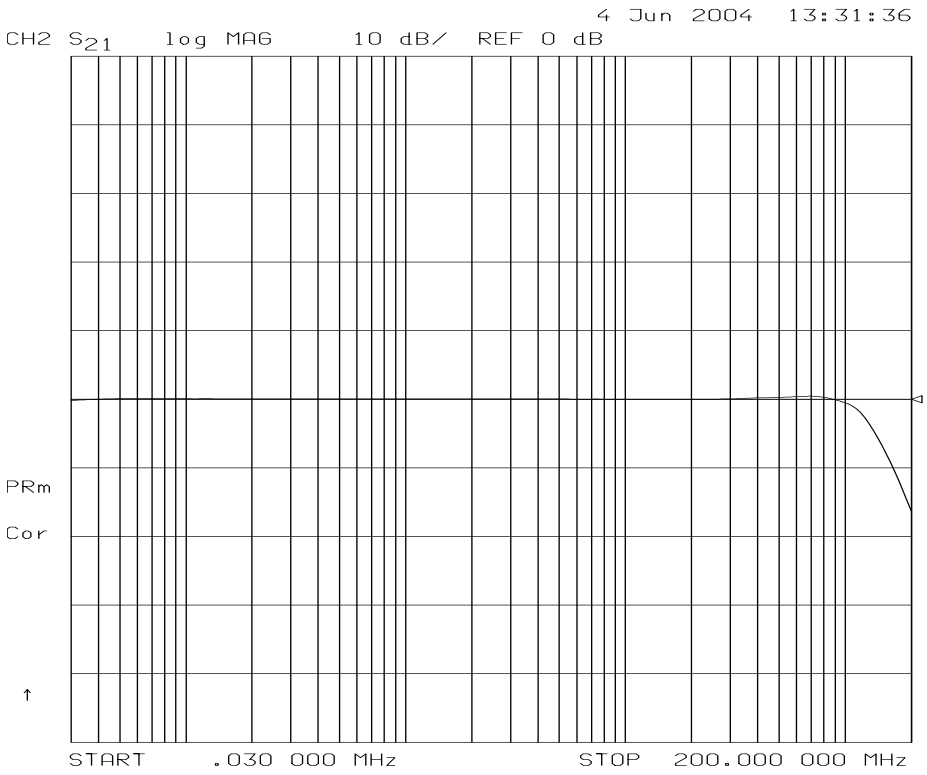

Figure 5: Typical frequency response

5.2 Typical delay

The delay was measured by creating a path, and connecting the calibrated analyzer to the input and the output of the path respectively. Figure 6. shows the plot of the typical delay of a created path versus the frequency.

5.3 Typical difference of delays between input groups

The first stage of the multiplexer has 3 multiplexer chips. The delay from the input terminals may differs, thus, different time delay could be introduced by the different signal paths. Figure 1 shows input grouping.

Figure 7 shows the difference of delay of input group A and input group B, vs. frequency. Figure 8 shows the difference of input group A and C.

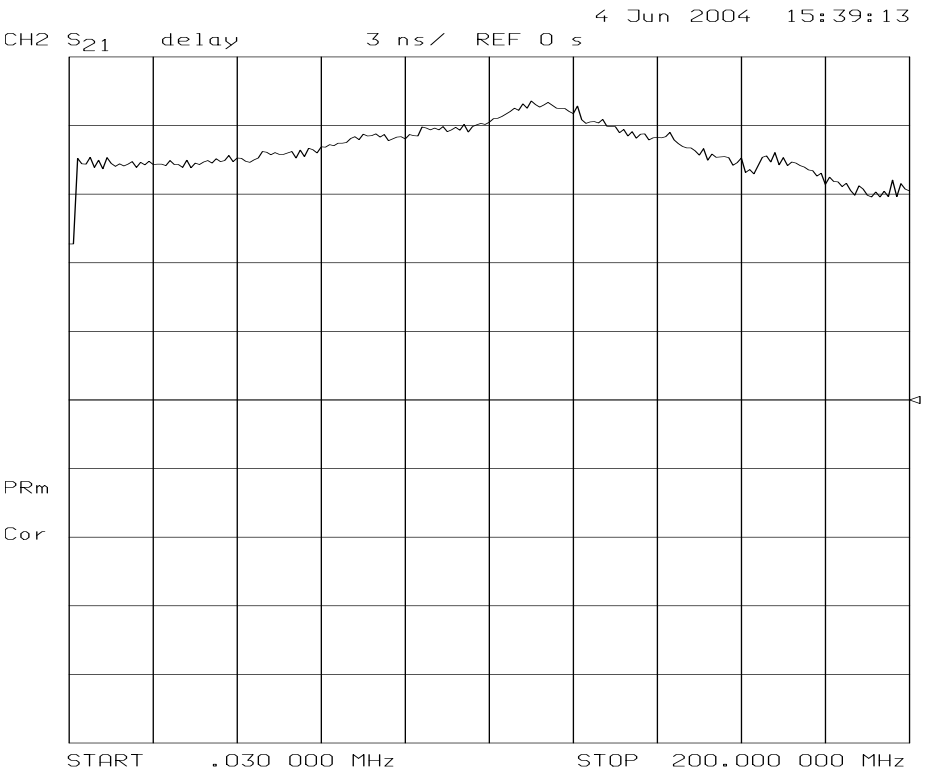

Figure 6: Typical delay

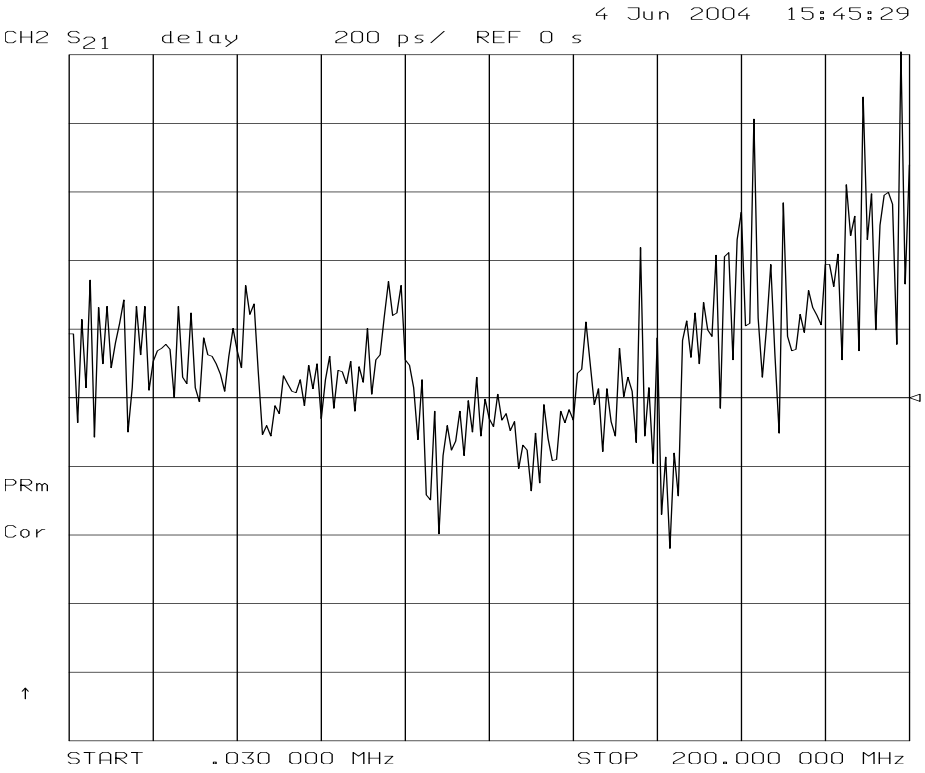

Figure 7: Delay difference between input group A and B

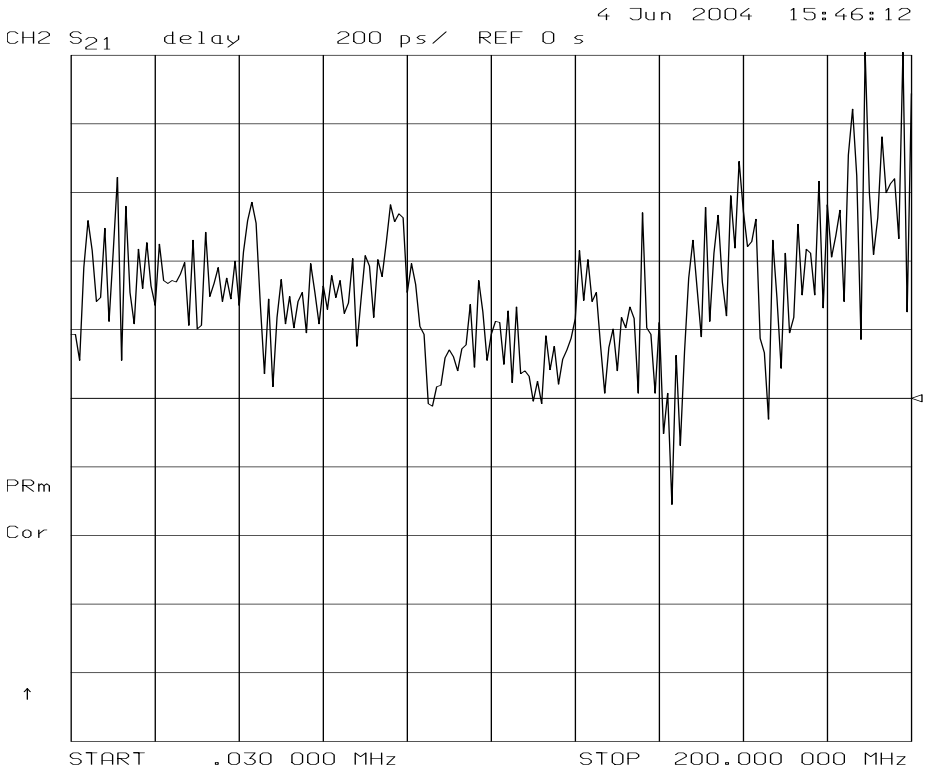

Figure 8: Delay difference between input group A and C

5.4 Typical difference of delays between output groups

Because of the lack of space, half of the output terminals of the multiplexer are connected by a 1ns coaxial cable to the PCB. Therefore there is some difference between delays of these two groups. These outputs are number 1,3,5 and 7. Figure 10 shows this delay vs. frequency.

5.5 Time domain

Figure 10 shows an impulse response. Note that the slope is not corresponding to the frequency response. The pulse generator had 10ns rising and falling time. This plot shows that there is no overshoot, and no other nonlinear distortion.

5.6 Crosstalk

Crosstalk have been measured by creating two paths, #1 and #2. The analyser was connected to input of path #1 and the output of path #2. All other inputs and outputs were terminated by 50Ω load. Result is shown on figure 11.

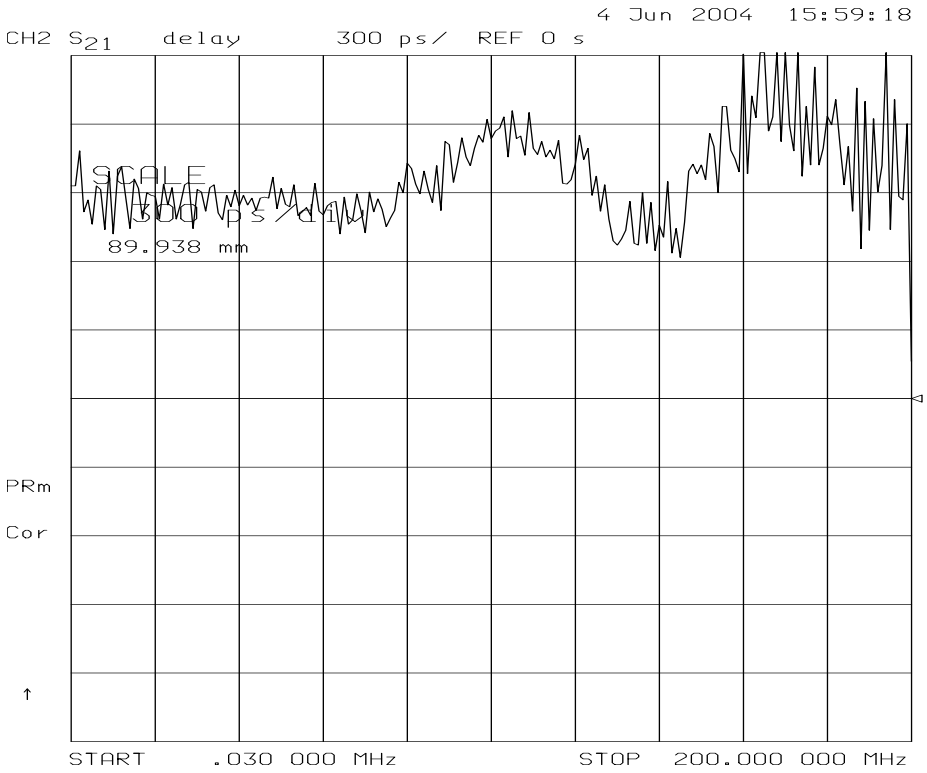

Figure 9: Delay difference between the two output groups

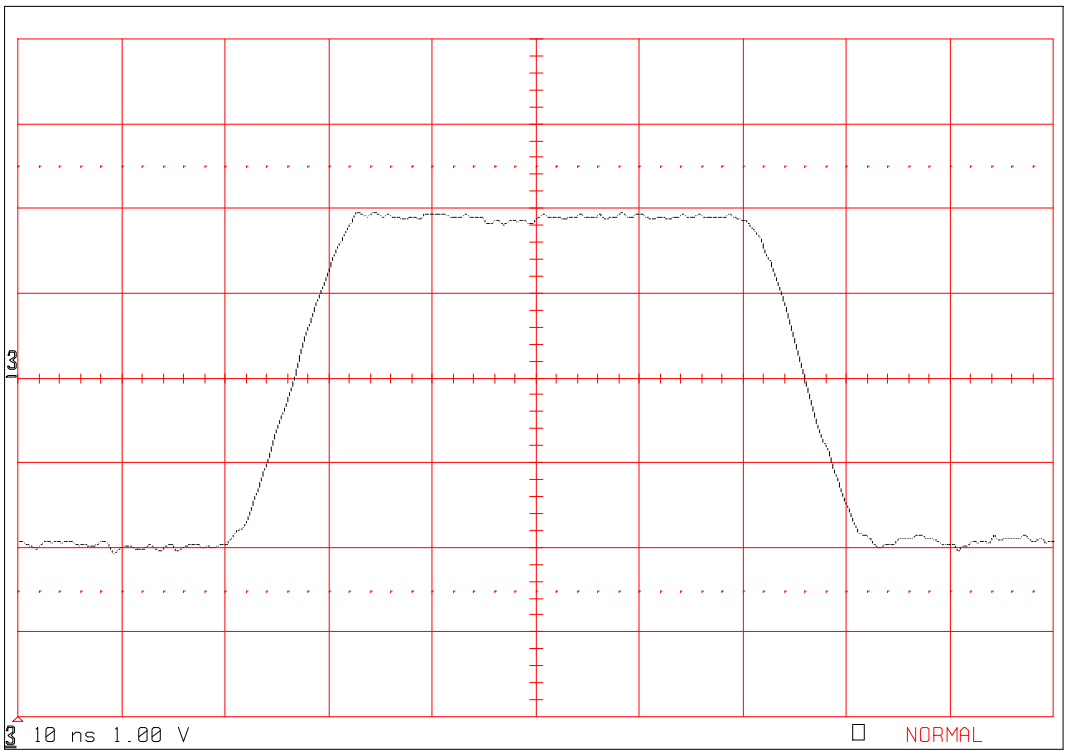

Figure 10: Impulse response

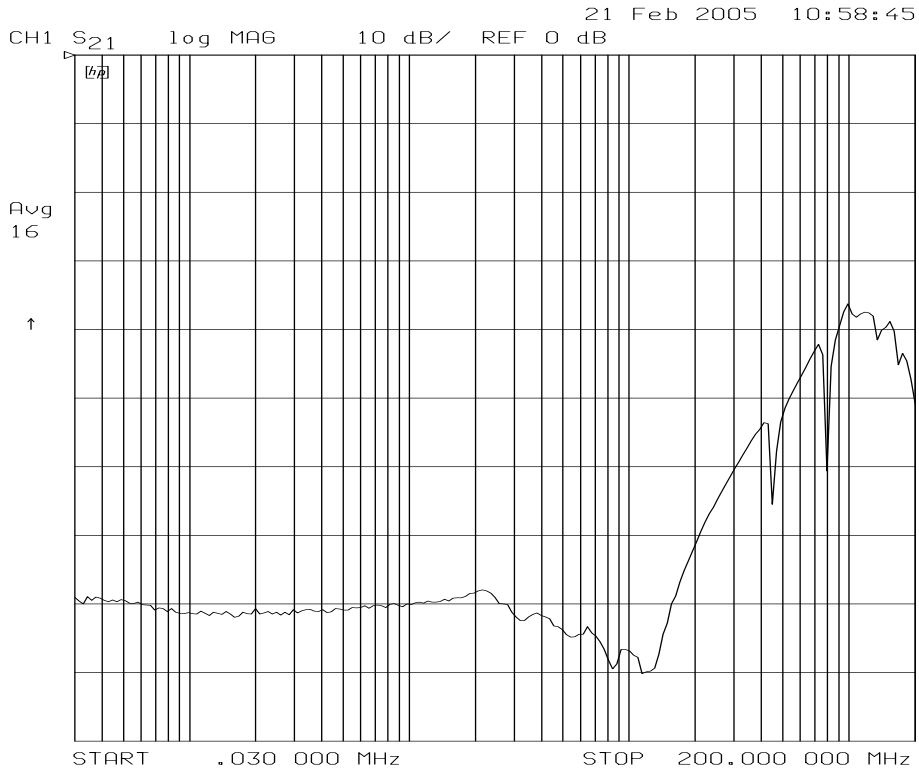

Figure 11: Crosstalk

6 Specifications

6.1 Interfaces

6.1.1 Input / output analogue interfaces

These are standard 50Ω female BNC connectors.

6.1.2 IEEE802.3

This is a standard 10Mbit/s, UTP Ethernet interface. Also known as 10BaseT. The connector is an RJ-45 female connector. See tabular 1 and figure 12 for pin out. For more information on the IEEE803.2 interface see [7].

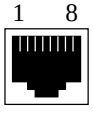

Figure 12: The layout of the IEEE802.3 interface.

6.1.3 V.24 (RS232C)

This interface conforms to the V.24/RS232C standards. It uses the standard DB9F connector. See figure 13 and tab. 2 for pin out. For more information on the V.24 interface, please refer

Pin number	function
1	TX+
2	TX-
3	RX+
6	RX-

Table 1: The pin out of the IEEE802.3 interface

to [6]. Please note that the RTS and CTS signals are connected together to implement pseudo hardware handshake.

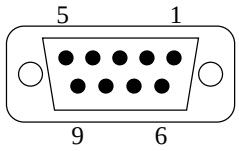

Figure 13: The layout of the V.24 interface

Pin number	Signal	Direction seen from the AMUX
1	N.C.	-
2	RXD	in
3	TXD	out
4	DTR	in
5	GND	-
6	DSR	out
7	RTS	out
8	CTS	in
9	N.C.	-

Table 2: The pin out of the V.24 interface

6.2 General specifications

See table 3 for general descriptions.

A Hardware errors

The schematic diagram and the PCB layout have an error. This error has been fixed on the PCB, but not on the schematic diagram, nor on the PCB layout diagram. If there will be further production of the AMUX, edit the schematic diagram and the PCB layout. In order to correct this problem, disconnect pin 2 of ST1, and connect it to pin 16 of IC1 ($+5V_{DC}$).

Description	value	unit
Input and output Impedance	50	Ω
Number of inputs	48	-
Number of outputs	8	-
Maximum input voltage	2,5	Vpp
Power consumption	50	W
Fuse rating	1	AT
Power supply voltage	230	V_{AC}
Dimensions	89x432(483)x397	VxHxD[mm]

Table 3: Specifications

B Hardware programming

In order to install or upgrade the software to the hardware, the following the process must be performed.

1. Connect the master host (usually a PC with the Zworld - Dynamic C software) to the V.24 (RS232C) interface with a straight (MODEM) serial cable.
2. Short the 2-1 pins of jumpers labelled S1 to S5.
3. Power up the AMUX, and the PC, and press the reset button. Start downloading the software as it is described in the manual of Dynamic C [5].
4. For normal operation, short 1-3 pins of S1 and S2 and remove the jumpers from S3,S4, and S5.

References

[1] EDA document EDA-00249

[2] ITU-T X.200 recommendation
INFORMATION TECHNOLOGY OPEN SYSTEMS INTERCONNECTION BASIC REFERENCE

[3] POSIX.1 IEEE Std 1003.1

[4] The Gimp Tool Kit widget set. Version GTK+-1.2.10 `http://www.gtk.org/`

[5] Zworld Dynamic C environment `http://www.zworld.com/`

[6] ITU-T V.24 recommendation
List of definitions for interchange circuits between data terminal equipment (DTE) and data circuit-terminating equipment (DCE)

[7] IEEE 802.3
Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications