DMA AMPLIFIERS SERIES

Third parties control protocol

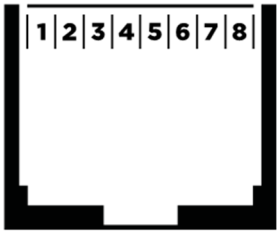
Protocollo di controllo terze parti

CONTENTS

SYSTEM ARCHITECTURE

In order to control the DMA series amplifiers through the communication protocol, the following devices are required:

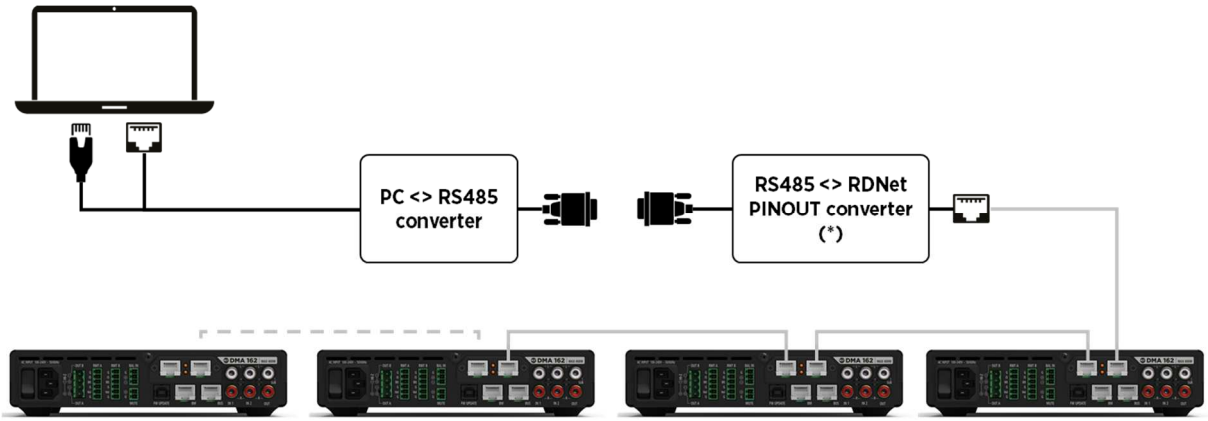
1. a PC <> RS485 converter, for example usb <> RS485;
2. an RS485 <> RDNET converter (on RJ45 connector). This device shall map on the RS485 connector (for example DB-9) the pinout necessary to communicate with the DMA device via RDNet protocol. The pinout is described in the table below:



PIN	INPUT	COLOUR
1	NC	OR/W
2	NC	OR
3	NC	GR/W
4	GND BUS	BL
5	GND BUS	BL/W
6	NC	GR
7	AIN 485	BR/W
8	BIN 485	BR

3. a RDNET BOARD DMA accessory (pn 12399048) for each device to be controlled.

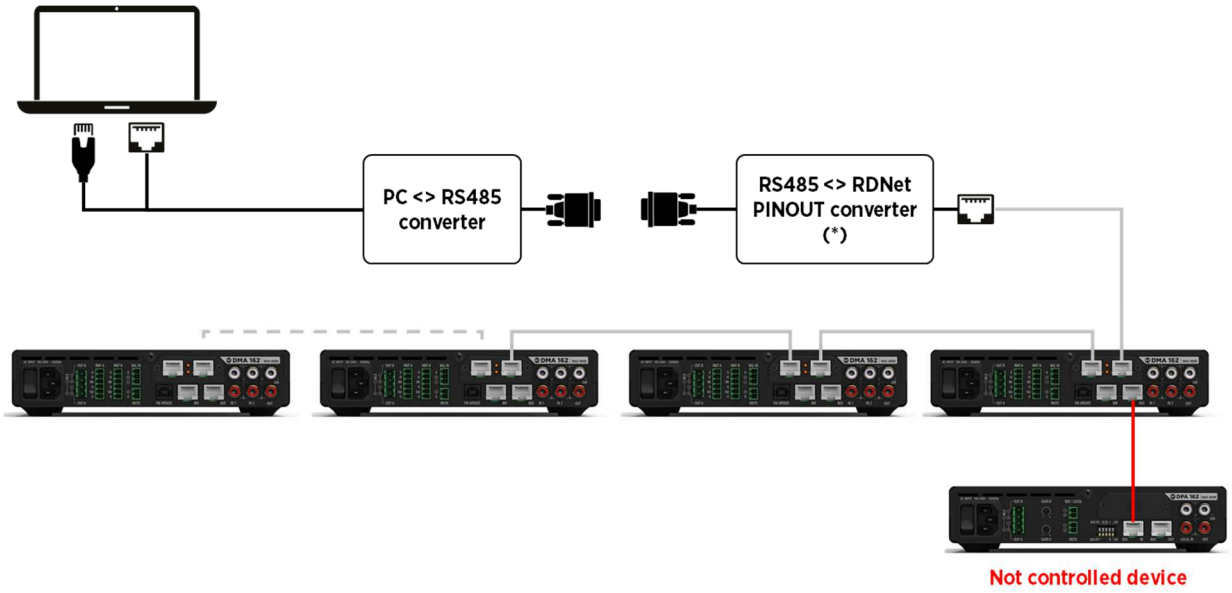
It is now possible to proceed with the following architecture.



Up to 32 devices can be connected, regardless of the model.

Note

Through this architecture it is possible to control only the devices connected via the RDNET BOARD DMA accessory, and therefore it is not possible to control the devices connected via BUS, as shown in the following figure.



TECHNICAL SPECIFICATIONS

The main technical characteristics of the communication network are listed in the table below.

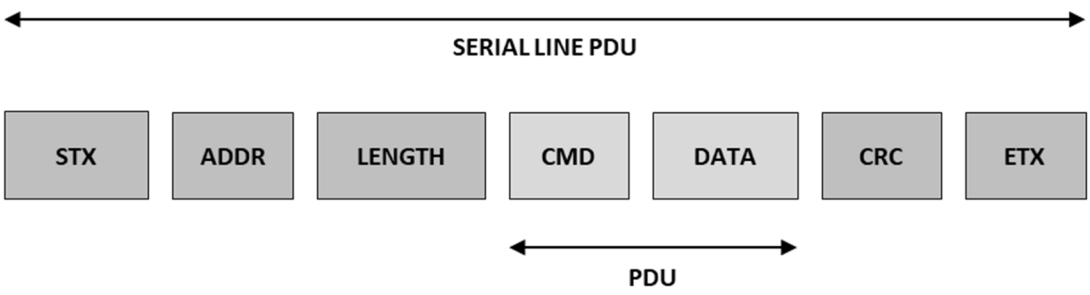
Status refresh rate	5 Hz ÷ 10 Hz
Data rate	115.2 kbit/s
Cyclic redundancy check	CRC-16
Serial transmission	8 bit No parity 1 stop
Bus maximum length	600 m (optimum) 1000 m (maximum)
Physical layer of communication between central and remote units	EIA-485
Dynamic assignment of addresses to remote units	SI
Addressing mode	Unicast/Broadcast

SYSTEM REQUIREMENTS

Firmware version	DMA 82 DMA 162	2.9
	DMA 162P	1.8

COMMUNICATION FRAME DESCRIPTION

The communication frame is made of a sequence of bytes ordered as follows:



The message frame begins with the **STX** symbol and ends with the **ETX** symbol. In this way, if the destination loses the boundaries of a frame, regains it when the next **STX** and **ETX** pair arrive.

However, there is a problem: in the transmission of binary data, the byte corresponding to the **STX** and **ETX** encoding can appear inside the frame, introducing confusion. To avoid this, the **byte stuffing** technique is used.

The above mentioned symbols are made up of copies of bytes all starting with the first byte **0X02** which acts as escape symbol.

STX and **ETX** symbols and **DATA** = 0x02 are encoded with the following bytes couple:

STX	(0x02, 0x02) Start transmission
ETX	(0x02, 0x03) Stop transmission
DATA 0x02	(0x02, 0x00) Generic 0x02 character

All other values are specified by their value in a single byte:

DATA = 0x00 ÷ 0x01 ÷ 0x00 ÷ 0x01

DATA = 0x03 ÷ 0xFF ÷ 0x03 ÷ 0xFF

After the **STX** field follows the **ADDR** field which identifies the remote slave unit. The range of slave addresses varies from 1 to 255. The master addresses a slave by entering the slave address in the **ADDR** field of the message. When the slave node returns its response, it enters its address in the **ADDR** field of the message to inform the master which slave it is responding to.

The range of addresses is divided as follows:

- The **ADDR** = 0x00 address corresponds to the default address of each slave.
- The address **ADDR** = 0XFF is reserved for broadcast messages that all slaves recognize.

After the **ADDR** field the **LENGTH** field comes which indicates the length in bytes of the **DATA** field.

The **CMD** field is used to indicate to the slave the type of action to be taken. This field uses 2 bytes and the range of codes varies from 0 to 65535.

The **CMD** field can be followed by the **DATA** field which contains the request and/or response data.

The **CRC** field or the **Cyclic Redundancy Code** contains the result of the redundancy check calculated on the transmitted message

The communication frame is therefore composed of the following bytes sequence:

STX[0]	0x02	START FRAME	
STX[1]	0x02	START FRAME	
ADDR	0x00 ÷ 0xFF	Slave address	
LENGTH	0x00 ÷ 0xFF	Length in bytes of the DATA queue	
CMD H	0x00 ÷ 0xFF	Command to be executed	MSByte
CMD L	0x00 ÷ 0xFF	Command to be executed	LSByte
DATA	0x00 ÷ 0xFF	DATA queue	
"	"	"	
"	"	"	
DATA	0x00 ÷ 0xFF	DATA queue	
CRC H	0x00 ÷ 0xFF	CRC16	MSByte
CRC L	0x00 ÷ 0xFF	CRC16	LSByte
ETX[0]	0x02	END FRAME	
ETX[0]	0x03	END FRAME	

The length of the frames is variable and is fixed at a maximum of 256 bytes. The payload for each communication frame is 8 bytes.

When the slave responds to the master, it uses the **CMD** to indicate both a correct response and a response to an error that has occurred (exception response). For a simple acknowledge response, the slave simply echoes the original **CMD**.

In the event of no response or incorrect response (mismatched control codes), the master makes some attempts to interrogate the same slave. The master can easily discriminate the type of response from the length, in fact: no data implies a positive response, a length equal to 1 identifies an error, otherwise the data relating to the query are reported.

CRC-16 CALCULATION

The **CRC** (Cyclic Redundancy Check) is made up of two bytes and is calculated by the transmitting device (both MASTER and SLAVE) which postpones it to the message. The receiving device recalculates the CRC when receiving the message and compares this value with the one received. If the two values are different, an error is reported.

The calculation of the CRC is started by preloading a 16-bit register with all bits at 1. Then the calculation of the CRC proceeds by inserting the bytes of the message to the current contents of the register. Only the 8 least significant bits in each character are used for CRC generation. The start (STX) and stop (ETX) characters do not apply for the CRC calculation.

During the CRC generation, an exclusive OR (XOR) logic operation is performed between each 8-bit character and the contents of the register.

Then the result is translated (shift operation) in the direction of the least significant bit (LSB) with an insertion of zero (zero filling) in the position of the most significant bit (MSB). The bit extracted in this way is then examined. If it is 1 then an XOR operation is performed between the contents of the register and a predetermined value which depends on the generator polynomial used. If it is 0, no XOR operation is performed.

The process is repeated until 8 translations of the register contents are made. Therefore a new character can be inserted in the register with an XOR operation between the character itself and the present content of the register.

The final content of the register, after all the characters of the message have been entered, is the CRC value.

The procedure for generating the CRC-16, used in the protocol, is summarized as follows:

1. Initialize a 16-bit register with the value 0xFFFF. This register is called the CRC register.
2. Execute the XOR operation in the CRC register between the first 8-bits of the message and the least significant byte of the register itself.
3. Left shift the CRC register by one bit, filling the most significant bit with zero.
4. If the extracted bit is zero, repeat point 3. If the extracted bit is one, execute the XOR operation between the contents of the register and the value 0xA001 which corresponds to the use of a generator polynomial $x^{16} + x^{15} + x^2 + 1$.
5. Repeat steps 3. and 4. until 8 translations have been performed to complete the CRC of the current byte of the message.
6. Repeat steps 2. to 5. for all subsequent bytes of the message.
7. The final content of the register is the CRC of the message.

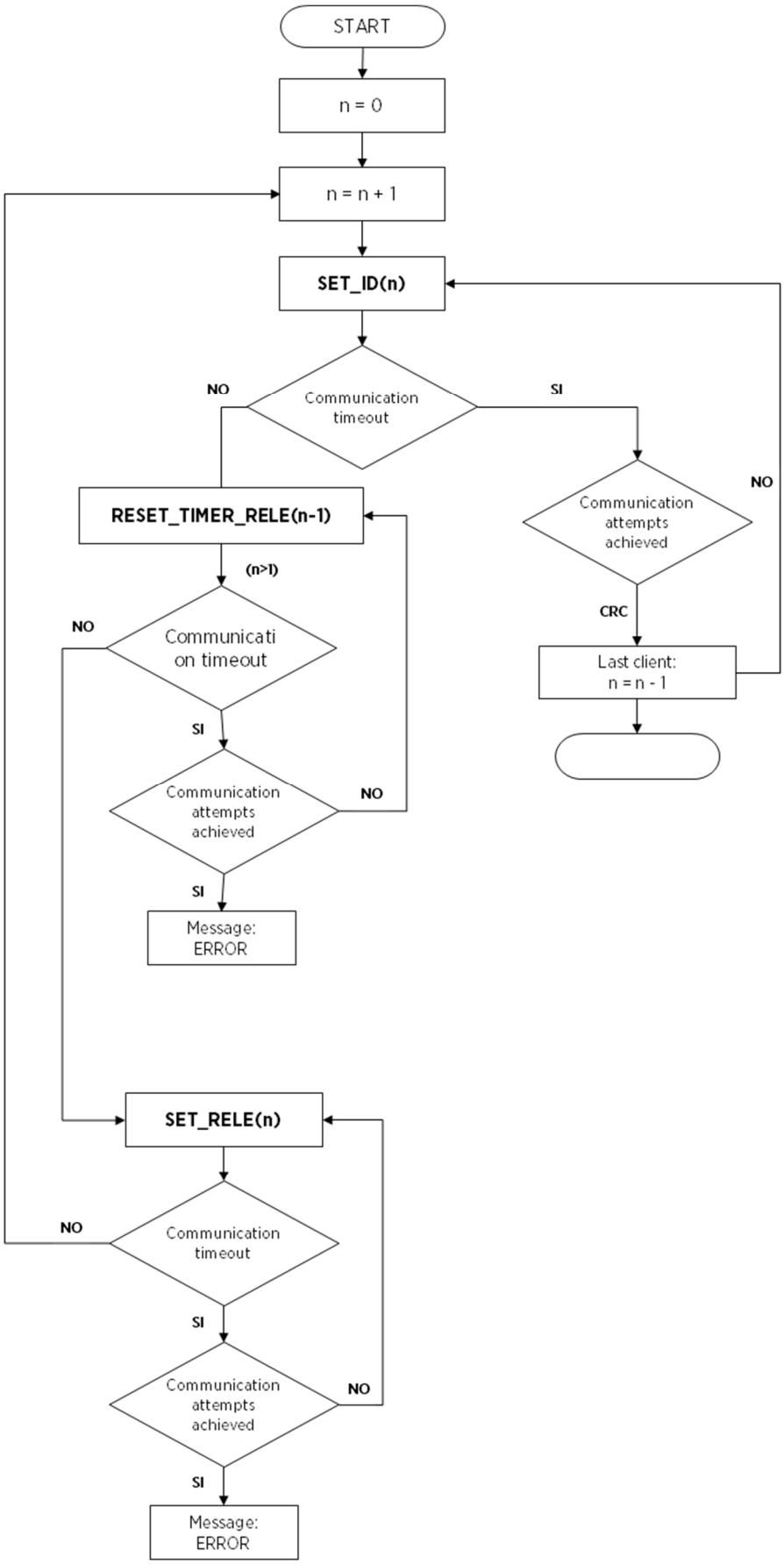
REGISTRATION SEQUENCE

To be able to logically distinguish the slaves from each other, it is necessary to assign a unique address to each node connected on the line. This assignment is made dynamically during the bus start-up phase.

Sequentially, starting from the remote unit physically closest to the master, an address is assigned for each slave. In this way the master gets the arrangement of the remote units, knowing their relative position.

When the system is turned on, the Slave units have their own bus isolation relay in the open position. In this configuration the slave units do not propagate the signal from the input communication connector to the output communication connector and only the first slave unit is connected to the line and correctly terminated.

Registration takes place according to the following block diagram:



Procedure description:

1. The Master unit sets ID_SLAVE = 1 as the first address to be assigned to the first Slave unit on the line. The Master unit sends the SET_ID command to assign address. This command has as the address of the recipient the value ADDRESS = 0, regardless of the Slave unit to which it is transmitted and as the only DATA [0] = ID_SLAVE = 1.

2. If the Slave unit receives this command correctly then it replaces the default address ID_SLAVE = 0 with the new one ID_SLAVE = 1 and responds with an ACK command to the Master unit. The procedure continues at step 3.

   If the Slave unit does not respond with an ACK command, the Master unit retries sending the command after a timeout period. If the expected number of attempts is exceeded, then there are no Slave units connected on the line (SET_ID = FALSE, SET_RELE = FALSE, RESET_TIMER_RELE = FALSE).

3. The Master unit sends the SET_RELE relay closing command to the current Slave unit with address ID_SLAVE = 1 to physically connect the next Slave unit on the line. In this communication, the address of the recipient is no longer the default one but is the value ADDRESS = ID_SLAVE = 1.

4. If the Slave unit receives this command correctly then it responds with an ACK command to the Master unit and, ONLY after the complete sending of the last byte of the ACK command, closes the relay. At this point the Slave unit starts a timer to control the relay closing timeout. If the Master unit does not stop counting within the time limit, the Slave unit opens the relay again. This timer is essential if there are no further slave units that terminate the line correctly, compromising any communication between the Master unit and the registered Slave units. The procedure continues at step 5.

   If the Slave unit does not respond with an ACK command, the Master unit retries sending the command after a timeout period. If the expected number of attempts is exceeded, an exception is raised in the registration procedure (SET_ID = TRUE, SET_RELE = FALSE, RESET_TIMER_RELE = FALSE).

5. The Master unit increases the ID_SLAVE address used in the previous step by a unit: ID_SLAVE (n + 1) = ID_SLAVE (n) + 1 = n + 1.

   The Master unit sends the SET_ID command for address assignment. This command has as the address of the recipient the value ADDRESS = 0, regardless of the Slave unit to which it is transmitted and as the only DATA [0] = ID_SLAVE (n + 1) = n + 1.

6. If the Slave unit receives this command correctly then it replaces the default address ID_SLAVE = 0 with the new one ID_SLAVE = n + 1 and responds with an ACK command to the Master unit. The procedure continues at step 7.

   If the Slave unit does not respond with an ACK command, the Master unit retries sending the command after a timeout period. If the expected number of attempts is exceeded, it means that there are no further Slave units on the line. In this situation, the slave unit with address SLAVE_ID = n, upon reaching the relay closing control timeout, will open the n-th relay again, restoring the line termination.

7. The Master unit sends the RESET_TIMER_RELE command to stop the relay timeout control timer to the Slave unit with address SLAVE_ID = n.

8. If the Slave unit with address SLAVE_ID = n correctly receives this command to stop the relay timeout control timer, it responds with an ACK command to the Master unit. The procedure continues at step 9.

If the Slave unit with address SLAVE_ID = n does not respond with an ACK command, the Master unit retries sending the command after a timeout period. If the expected number of attempts is exceeded, an exception is raised in the registration procedure (SET_ID = TRUE, SET_RELE = TRUE, RESET_TIMER_RELE = FALSE).

9. The Master unit sends the SET_RELE relay closing command to the current Slave unit with address ID_SLAVE = n + 1 to physically connect the next Slave unit on the line. In this communication, the address of the recipient is no longer the default one but is the value ADDRESS = ID_SLAVE = n + 1.
10. If the Slave unit correctly receives this command to close the relay, it responds with an ACK command to the Master unit and, ONLY after sending the complete last byte of the ACK command, closes the relay. At this point the Slave unit starts a timer to control the relay closing timeout. The procedure continues at point 5. If the Slave unit does not respond with an ACK command, the Master unit retries sending the command after a timeout period. If the expected number of attempts is exceeded, an exception is raised in the registration procedure (SET_ID = TRUE, SET_RELE = FALSE, RESET_TIMER_RELE = FALSE).

Points 5 to 10 are iterated.

The registration procedure is finished.

KEEPING COMMUNICATION

To keep the communication, it is necessary to poll each device on the bus requesting its status with a refresh rate higher than 0.1Hz.

DE-REGISTRATION SEQUENCE

To be able to change the number of slave units of a subnet or add or remove SLAVE units, a bus de-registration operation is required.

This operation simply consists in waiting for the communication timeout (10 seconds), after which the relay is closed in the termination position and the reinitialization of the device with ID_SLAVE = 0.

PROTOCOL COMMANDS

SLAVE STATUS REQUEST

Description: sends the status request to the slave.

The slave responds with a sequence of N bytes describing the status of the device.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Slave status request	ID	0	1	-
ACK sending	ID	10	1	<DATA[0..10]>
No ACK sending	ID	1	1	ERROR CODE

PAYLOAD STATUS

BYTE 1 - error flags:
- bit 1: reserved
- bit 2: temperature error flag
- bit 3: dsp error flag
- bit 4: generic error flag
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

BYTE 2 - mute and peak flags:
- bit 1: mute flag channel 1
- bit 2: peak flag channel 1
- bit 3: mute flag channel 2
- bit 4: peak flag channel 2
- bit 5: reserved
- bit 6: reserved
- bit 7: reserved
- bit 8: reserved

BYTE 3 - channel 1 signal value

BYTE 4 - channel 2 signal value

BYTE 5 - reserved

BYTE 6 - reserved

BYTE 7 - live flags:

- bit 1: paging on air (1 on air, 0 no active)
- bit 2: OUT A busy
- bit 3: OUT B busy
- bit 4: reserved
- bit 5: reserved
- bit 6: reserved
- bit 7: not used
- bit 8: not used

BYTE 8 - reserved

BYTE 9 - reserved

BYTE 10 - reserved

ID SLAVE SENDING

Description: assigns a new address to the slave (The starting address is always 0).

The slave responds with the sequence ACK or No ACK.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
ID Slave sending	0	1	2	ID
ACK sending	ID	0	2	-
No ACK sending	ID	1	2	ERROR CODE

SLAVE RELÈ OPENING

Description: forces the relay to open (bypass position INPUT to LINK, this slave will no longer be terminated).

The slave responds with the sequence ACK or No ACK before switching.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Slave relè opening	ID	0	3	-

ACK sending	ID	0	3	-
No ACK sending	ID	1	3	ERROR CODE

SLAVE RELÈ CLOSING

Description: forces the closing of the relay, the LINK line is deactivated and the slave terminates the communication channel.

The slave responds with the sequence ACK or No ACK before switching.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Slave relè closing	ID	0	4	-
ACK sending	ID	0	4	-
No ACK sending	ID	1	4	ERROR CODE

STOP RELÈ TIMER

Description: stops the countdown set in 5 seconds after which the termination on the slave would be set (relay closure).

The slave responds with the sequence ACK or No ACK.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Stop relè timer	ID	0	5	-
ACK sending	ID	0	5	-
No ACK sending	ID	1	5	ERROR CODE

DMA 82 / DMA 162 COMMANDS

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
COMMAND	ID	0	CODE	-
ACK sending	ID	N	CODE	<DATA[0..10]>
No ACK sending	ID	1	CODE	ERROR CODE

PAYLOAD STATUS

INIT COMMANDS

CODE 0x11 **INIT GENERAL**

BYTE 1 - fw revision

BYTE 2 - general flags:

- bit 1: stereo/mono flag OUT A e B
- bit 2: reserved
- bit 3: bridge mode flag OUT A e B
- bit 4: reserved
- bit 5: reserved
- bit 6: reserved
- bit 7: reserved
- bit 8: reserved

BYTE 3 to 8 - reserved

CODE 0x12 **INIT OUTPUT**

BYTE 1 to 4 - volume OUT A

BYTE 5 - OUT A channel flags

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used

- bit 8: not used

BYTE 6 to 8 - reserved

BYTE 9 - input code OUT A

BYTE 10 to 31 - reserved

BYTE 32 to 35 - volume OUT B

BYTE 36 - OUT B flags:

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

BYTE 37 to 39 - reserved

BYTE 40 - input code OUT B

BYTE 41 to 124 - reserved

CODE 0x13 INIT INPUT

BYTE 1 to 4 - stereo volume IN1

BYTE 5 to 8 - mono volume IN1 L

BYTE 9 to 12 - mono volume IN1 R

BYTE 13 to 48 - reserved

BYTE 49 to 52 - stereo volume IN2

BYTE 53 to 56 - mono volume IN2 L

BYTE 57 to 60 - mono volume IN2 R

BYTE 61 to 96 - reserved

BYTE 97 to 100 - stereo volume EXT

BYTE 101 to 104 - mono volume EXT L

BYTE 105 to 108 - mono volume EXT R

BYTE 109 to 144 - reserved

BYTE 145 to 148 - mono volume BAL IN

BYTE 149 to 160 - reserved

BYTE 161 to 164 - mono volume PAGING

BYTE 165 to 176 - reserved

OUTPUTS COMMANDS

CODE 0x31 SET VOLUME OUTPUT

BYTE 1 - output code

BYTE 2 to 5 - gain value

CODE 0x32 SET GENERAL VOLUME OUTPUT

BYTE 1 to 4 - gain value OUT A

BYTE 5 to 8 - gain value OUT B

BYTE 9 to 12 - gain value OUT C

BYTE 13 to 16 - gain value OUT D

CODE 0x35 ROUTING

BYTE 1 - output code

BYTE 2 - input code

CODE 0x3A SET MUTE OUTPUT

BYTE 1 - output code

BYTE 2 - mute flag:

- bit 1: mute/unmute flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODE 0x3B SET MUTE GENERALE OUTPUT

BYTE 1 - mute flag:

- bit 1: mute/unmute flag OUT A (1/0)
- bit 2: mute/unmute flag OUT B (1/0)
- bit 3: mute/unmute flag OUT C (1/0)
- bit 4: mute/unmute flag OUT D (1/0)
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODE 0x3E **GENERAL ROUTING**

BYTE 1 - input code

CODE 0x3F **SET VOLUME STEREO/BRIDGE OUTPUT**

BYTE 1 - output code (1 for the couple OUT A-B)

BYTE 2 to 5 - gain value output OUT A

BYTE 6 to 9 - gain value output OUT B

CODE 0x40 **SET MUTE STEREO/BRIDGE OUTPUT**

BYTE 1 - output code (1 for the couple OUT A-B)

BYTE 2 - mute flag:

- bit 1: mute/unmute flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODE 0x41 **ROUTING STEREO**

BYTE 1 - output channels code (1 for the couple OUT A-B)

BYTE 2 - input code

CODE 0x42 **GENERAL VOLUME INPUT**

BYTE 1 to 4	- gain value IN1 L
BYTE 5 to 8	- gain value IN1 R
BYTE 9 to 12	- gain value IN2 L
BYTE 13 to 16	- gain value IN2 R
BYTE 17 to 20	- gain value EXT L
BYTE 21 to 24	- gain value EXT R
BYTE 25 to 28	- gain value BAL IN
BYTE 29 to 32	- gain value PAGING

INPUTS COMMANDS

CODE 0x51 **SET VOLUME INPUT MONO**

BYTE 1	- input code
BYTE 2 to 5	- gain value

OUPUT CODES

OUT A	0x01
OUT B	0x02

INPUT CODES

IN1 L+R	0x01
IN2 L+R	0x02
EXT L+R	0x03
BAL IN	0x04
PAGING	0x05
IN1 L	0x06
IN1 R	0x07
IN2 L	0x08
IN2 R	0x09
EXT L	0x0A
EXT L	0x0B

DMA 162P COMMANDS

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
COMMAND	ID	0	CODE	-
ACK sending	ID	N	CODE	<DATA[0..10]>
No ACK sending	ID	1	CODE	ERROR CODE

PAYLOAD STATUS

INIT COMMANDS

CODE 0x11 **INIT GENERAL**

BYTE 1 - fw revision

BYTE 2 - general flags:

- bit 1: stereo/mono flag OUT A e B
- bit 2: reserved
- bit 3: bridge mode flag OUT A e B
- bit 4: reserved
- bit 5: reserved
- bit 6: reserved
- bit 7: reserved
- bit 8: reserved

BYTE 3 to 8 - reserved

CODE 0x12 **INIT OUTPUT**

BYTE 1 to 4 - volume OUT A

BYTE 5 - OUT A channel flags

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used

- bit 8: not used

BYTE 6 to 8 - reserved

BYTE 9 - input code OUT A

BYTE 10 to 31 - reserved

BYTE 32 to 35 - volume OUT B

BYTE 36 - OUT B flags:

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

BYTE 37 to 39 - reserved

BYTE 40 - input code OUT B

BYTE 41 to 124 - reserved

CODE 0x15 INIT INPUT

BYTE 1 to 4 - stereo volume IN1

BYTE 5 to 8 - mono volume IN1 L

BYTE 9 to 12 - mono volume IN1 R

BYTE 13 to 48 - reserved

BYTE 49 to 52 - mono volume CH1

BYTE 53 to 64 - reserved

BYTE 65 to 68 - mono volume CH2

BYTE 69 to 80 - reserved

BYTE 81 to 84 - mono volume CH3

BYTE 85 to 96 - reserved

BYTE 97 to 100 - mono volume CH4

BYTE 101 to 112 - reserved

OUTPUTS COMMANDS

CODE 0x31 **SET VOLUME OUTPUT**

BYTE 1 - output code

BYTE 2 to 5 - gain value

CODE 0x35 **ROUTING**

BYTE 1 - output code

BYTE 2 - input code

CODE 0x3A **SET MUTE OUTPUT**

BYTE 1 - output code

BYTE 2 - mute flag:

- bit 1: mute/unmute flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

INPUTS COMMANDS

CODE 0x51 **SET VOLUME INPUT MONO**

BYTE 1 - input code

BYTE 2 to 5 - gain value

GENERAL COMMANDS

CODE 0x26 **SET BUS LOCAL MODE**

BYTE 1 - BUS/LOCAL

- bit 1: local/bus mode flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used

- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

OUPUT CODES

OUT A	0x01
OUT B	0x02

INPUT CODES

CH1	0x01
CH2	0x02
CH3_IN1_R (*)	0x03
CH4_IN1_L (*)	0x04
IN1 L+R	0x05

(*) Bus channel 3 and 4 are physically overlaid with the inputs RCA1L and RCA1R. To interact with one of these inputs, make sure to be in the correct mode, local for RCA or bus for channels.

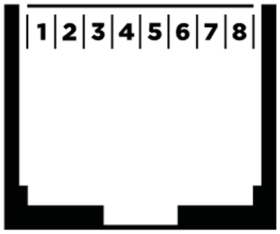
Notes

1. When the amplifier is in local mode and stereo mode, to change the volume / mute the output channels, send the Set Volume Output / Set Mute Output command to both channel A and channel B.
2. When the amplifier is in local or bus mode and in bridge mode, to change the volume / mute the output channels, send the Set Volume Output / Set Mute Output command only to channel A.

ARCHITETTURA DEL SISTEMA

Per poter controllare gli amplificatori della serie DMA attraverso il protocollo di comunicazione sono necessari i seguenti dispositivi:

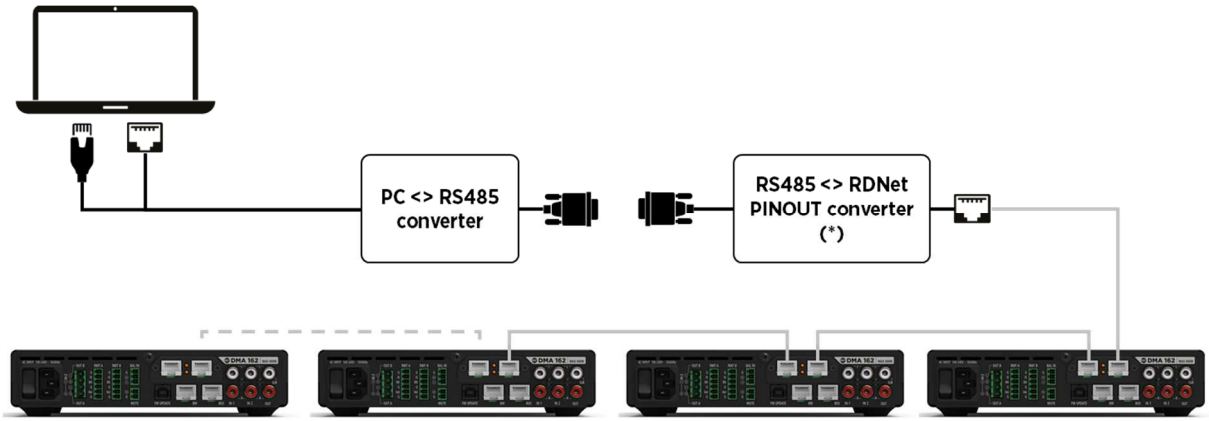
4. un convertitore PC <> RS485, ad esempio usb <> RS485;
5. un convertitore RS485 <> RDNET (su connettore RJ45). Questo dispositivo deve mappare sul connettore RS485 (ad esempio DB-9) il pinout necessario per dialogare con il dispositivo DMA tramite protocollo RDNet. Il pinout è descritto nella tabella che segue:



PIN	INPUT	COLOUR
1	NC	OR/W
2	NC	OR
3	NC	GR/W
4	GND BUS	BL
5	GND BUS	BL/W
6	NC	GR
7	AIN 485	BR/W
8	BIN 485	BR

6. una scheda accessoria RDNET BOARD DMA (pn 12399048) per ciascun dispositivo che si vuole controllare.

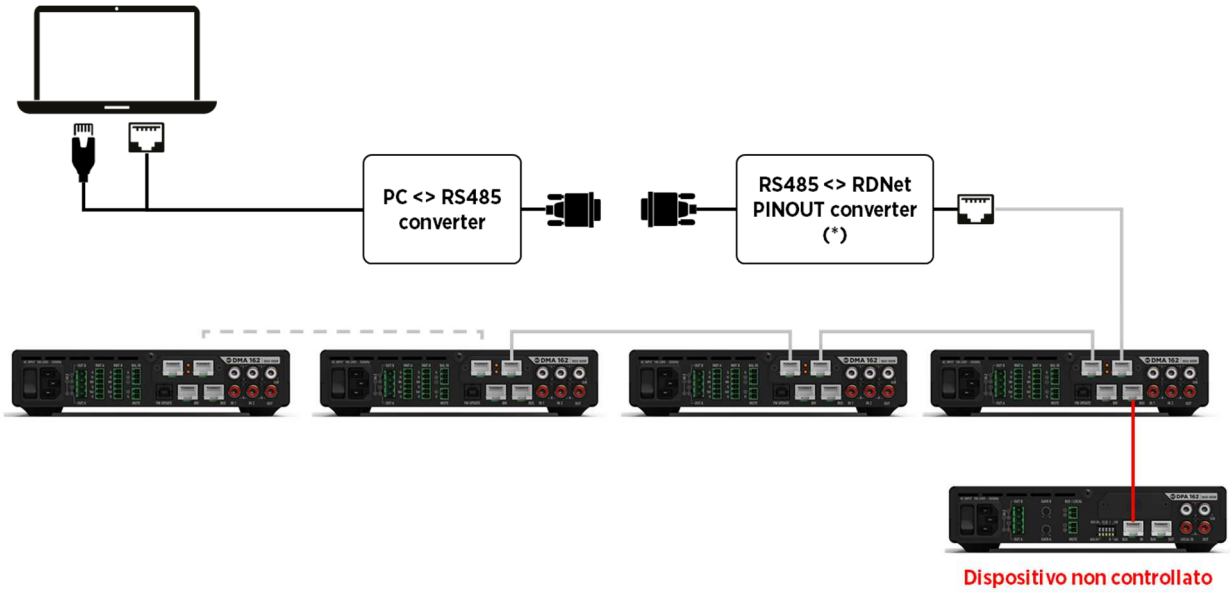
Si può quindi procedere all'allestimento della seguente architettura.



È possibile collegare fino a 32 dispositivi, indipendentemente dal modello.

Nota

Tramite questa architettura è possibile controllare solo i dispositivi collegati tramite scheda accessoria RDNET BOARD DMA, e non è quindi possibile controllare i dispositivi collegati tramite BUS, come mostrato nella figura seguente.



SPECIFICHE TECNICHE

Di seguito sono riportate le principali caratteristiche tecniche del network di comunicazione.

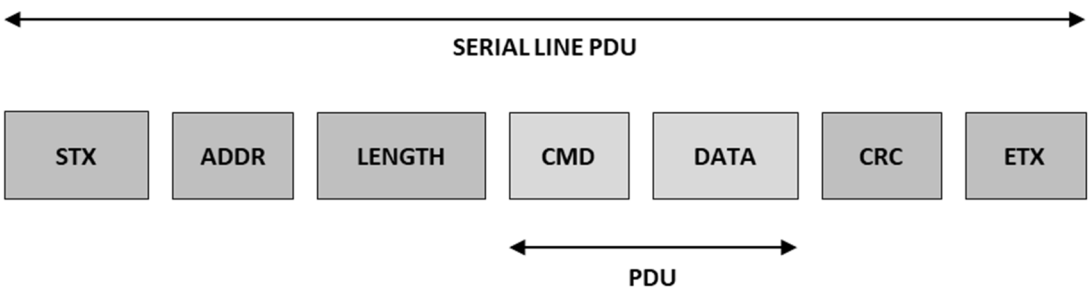
Refresh rate dello status	5 Hz ÷ 10 Hz
Data rate	115.2 kbit/s
Controllo d'errore comunicazione	CRC-16
Trasmissione seriale	8 bit No parity 1 stop
Lunghezza massima bus	600 m (optimum) 1000 m (maximum)
Layer fisico di comunicazione tra unità centrale ed unità remote	EIA-485
Assegnazione dinamica degli indirizzi alle unità remote	SI
Modalità di indirizzamento	Unicast/Broadcast

REQUISITI DI SISTEMA

Revisione firmware minima	DMA 82 DMA 162	2.9
	DMA 162P	1.8

DESCRIZIONE DEL FRAME DI COMUNICAZIONE

Il frame di comunicazione è composto da una sequenza di byte così ordinati:



Il frame di messaggio inizia con il simbolo di **STX** e termina con il simbolo di **ETX**. In questo modo, se la destinazione perde traccia dei confini di un frame la riacquista all'arrivo della prossima coppia **STX** e **ETX**.

Esiste però un problema: nella trasmissione di dati binari, il byte corrispondente alla codifica di **STX** e di **ETX** può apparire dentro il frame, confondendo le cose.

Per evitare questa evenienza, si utilizza la tecnica di **byte stuffing**.

I simboli suddetti sono costituiti da copie di byte che iniziano tutte con il primo byte 0X02 che funziona da simbolo di escape.

Il simbolo di **STX**, di **ETX** ed il **DATA** = 0x02 sono codificati con una coppia di byte nel modo seguente:

STX	(0x02, 0x02) Start transmission
ETX	(0x02, 0x03) Stop transmission
DATO 0x02	(0x02, 0x00) Carattere 0x02 generico

Tutti gli altri valori al di fuori dei suddetti vengono specificati dal loro valore in un solo byte:

DATA = 0x00 ÷ 0x01 ÷ 0x00 ÷ 0x01

DATA = 0x03 ÷ 0xFF ÷ 0x03 ÷ 0xFF

Dopo il campo di **STX** segue il campo **ADDR** che identifica l'unità remota di slave. L'intervallo degli indirizzi di slave varia da 1 a 255. Il master indirizza uno slave inserendo l'indirizzo di slave nel campo **ADDR** del messaggio. Quando il nodo di slave ritorna la sua risposta, inserisce il proprio indirizzo nel campo **ADDR** del messaggio per informare il master su quale slave sta rispondendo.

L'intervallo di indirizzi è stato così suddiviso:

- L'indirizzo **ADDR** = 0x00 corrisponde a quello di default di ogni slave.
- L'indirizzo **ADDR** = 0XFF è riservato per i messaggi di broadcast che tutti gli slave riconoscono.

Dopo il campo di **ADDR** segue il campo di **LENGTH** che indica la lunghezza in byte del campo **DATA**.

Il campo **CMD** serve ad indicare allo slave il tipo di azione da intraprendere. Tale campo utilizza 2 byte e l'intervallo dei codici varia da 0 a 65535.

Il campo **CMD** può essere seguito dal campo **DATA** che contiene i dati di richiesta e/o di risposta.

Il campo **CRC** ovvero il **Cyclic Redundancy Code** contiene il risultato del controllo di ridondanza calcolato sul messaggio trasmesso.

Il frame di comunicazione è quindi composto da una sequenza di byte così ordinati:

STX[0]	0x02	START FRAME	
STX[1]	0x02	START FRAME	
ADDR	0x00 ÷ 0xFF	Indirizzo dello Slave	
LENGTH	0x00 ÷ 0xFF	Lunghezza in byte della coda di DATI	
CMD H	0x00 ÷ 0xFF	Comando da eseguire	MSByte
CMD L	0x00 ÷ 0xFF	Comando da eseguire	LSByte
DATA	0x00 ÷ 0xFF	Coda dei dati	
"	"	"	
"	"	"	
DATA	0x00 ÷ 0xFF	Coda dei dati	
CRC H	0x00 ÷ 0xFF	CRC16	MSByte
CRC L	0x00 ÷ 0xFF	CRC16	LSByte
ETX[0]	0x02	END FRAME	
ETX[0]	0x03	END FRAME	

La lunghezza dei frame è variabile ed è fissata ad un massimo di 256 byte. Il payload per ogni frame di comunicazione è pari a 8 byte.

Quando lo slave risponde al master utilizza il **CMD** per indicare sia un risposta corretta sia una risposta ad un errore accaduto (exception response). Per un risposta di semplice acknowledge, lo slave fa semplicemente l'eco del **CMD** originale.

Il master, nel caso di mancata risposta o di risposta errata (codici di controllo non corrispondenti) effettua di seguito alcuni tentativi di interrogazione dello stesso slave. Il master discrimina facilmente il tipo di risposta dalla lunghezza, infatti: nessun dato implica una risposta positiva, una lunghezza pari ad 1 identifica un errore, diversamente vengono riportati i dati relativi all'interrogazione.

CALCOLO DEL CRC-16

Il controllo di ridondanza ciclica **CRC** è composto da due byte ed è calcolato dal dispositivo trasmittente (sia MASTER che SLAVE) che lo pospone al messaggio. Il dispositivo ricevente ricalcola il CRC durante la ricezione del messaggio e compara tale valore con quello ricevuto. Se i due valori non sono uguali viene segnalato un errore.

Il calcolo del CRC è iniziato precaricando un registro a 16 bit con tutti i bit a 1. Quindi il computo del CRC procede inserendo via via i byte del messaggio al contenuto corrente del registro. Solo gli 8 bit meno significativi in ogni carattere sono usati per la generazione del CRC. I caratteri di start (STX) e di stop (ETX) non si applicano per il calcolo del CRC.

Durante la generazione del CRC, si esegue una operazione logica di OR esclusivo (XOR) tra ogni carattere ad 8 bit ed il contenuto del registro.

Quindi il risultato è traslato (operazione di shift) in direzione del bit meno significativo (LSB) con un inserimento dello zero (zero filling) in posizione del bit più significativo (MSB). Il bit estratto in questo modo viene quindi esaminato. Se vale 1 allora si esegue una operazione di XOR tra il contenuto del registro ed un valore prefissato che dipende dal polinomio generatore utilizzato. Se vale 0 non viene intrapresa nessuna operazione di XOR.

Il processo è ripetuto fin quando vengano effettuati 8 traslazioni del contenuto del registro. Quindi un nuovo carattere può essere inserito nel registro con una operazione di XOR tra il carattere stesso ed il contenuto presente del registro.

Il contenuto finale del registro, dopo che sono stati inseriti tutti i caratteri del messaggio, è il valore di CRC.

La procedura per la generazione del CRC-16, utilizzato nel protocollo, è riassunta nel modo seguente:

8. Inizializzare un registro a 16-bit con il valore 0xFFFF. Tale registro è nominato registro CRC.
9. Eseguire nel registro CRC, l'operazione di XOR tra i primi 8-bit del messaggio ed il byte meno significativo del registro stesso.
10. Traslare a sinistra il registro CRC di un bit, riempiendo il bit più significativo con zero.
11. Se il bit estratto è zero ripetere il punto 3. Se il bit estratto è uno eseguire l'operazione di XOR tra il contenuto del registro ed il valore 0xA001 che corrisponde all'utilizzo di un polinomio generatore $x^{16}+x^{15}+x^{2}+1$.
12. Ripetere i punti 3. e 4. fino ad eseguire 8 traslazioni per completare il CRC del byte corrente del messaggio.
13. Ripetere i punti dal 2. al 5. per tutti i successivi byte del messaggio.
14. Il contenuto finale del registro è il CRC del messaggio.

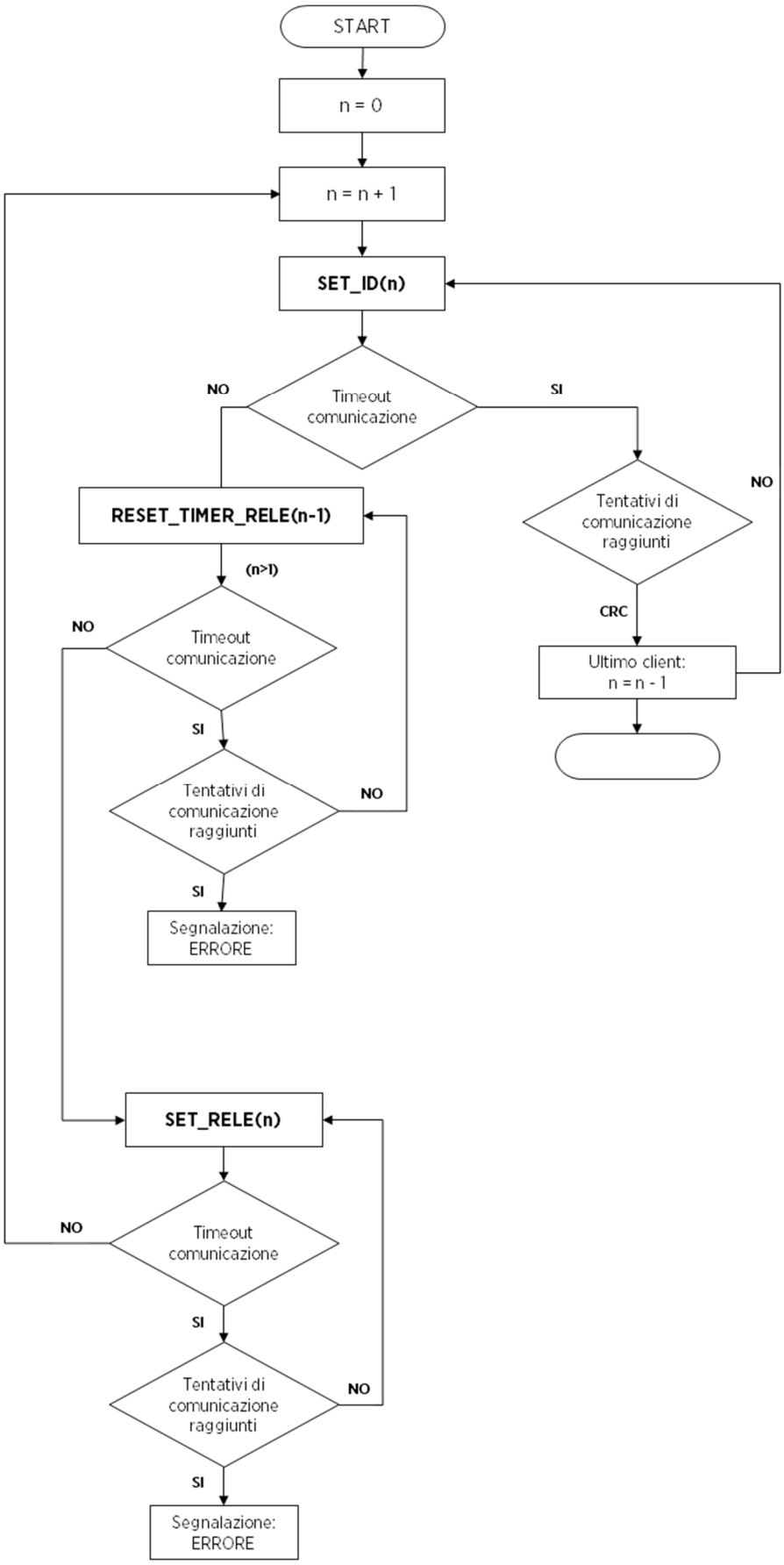
SEQUENZA DI REGISTRAZIONE

Per poter distinguere logicamente gli slave tra loro, è necessario assegnare un indirizzo univoco a ciascuno nodo collegato sulla linea. Tale assegnazione viene fatta in modo dinamico in fase di start-up del bus.

In modo sequenziale, a partire dalla unità remota fisicamente più vicina al master, viene assegnato un indirizzo per ogni slave. In tal modo il master ha la disposizione delle unità remote, conoscendone la loro posizione relativa.

All'accensione del sistema, le unità Slave hanno il proprio relè di sezionamento del bus in posizione aperta. In questa configurazione le unità slave non propagano il segnale dal connettore di comunicazione di ingresso a quello di comunicazione di uscita e solo la prima unità slave è connessa alla linea e correttamente terminata.

La registrazione avviene secondo il seguente schema a blocchi:



Andando a descrivere la procedura:

11. L'unità Master imposta come primo indirizzo da assegnare alla prima unità Slave della linea ID_SLAVE = 1. L'unità Master invia il comando SET_ID di assegnazione indirizzo. Tale comando ha come indirizzo del destinatario il valore ADDRESS = 0, indipendentemente dalla unità Slave a cui viene trasmesso e come unico DATA[0] = ID_SLAVE = 1.

12. Se l'unità Slave riceve correttamente tale comando allora sostituisce l'indirizzo di default ID_SLAVE = 0 con quello nuovo ID_SLAVE = 1 e risponde con un comando di ACK all'unità Master. La procedura continua al punto 3.

    Se l'unità Slave non risponde con un comando di ACK, l'unità Master ritenta l'invio del comando dopo un periodo di timeout. Qualora si superi il numero previsto di tentativi allora non vi sono unità Slave collegate sulla linea (SET_ID = FALSE, SET_RELE = FALSE, RESET_TIMER_RELE = FALSE).

13. L'unità Master invia il comando SET_RELE di chiusura relè all'unità Slave corrente di indirizzo ID_SLAVE = 1 per collegare fisicamente la successiva unita Slave della linea. In tale comunicazione l'indirizzo del destinatario non è più quello di default ma è il valore ADDRESS = ID_SLAVE = 1.

14. Se l'unità Slave riceve correttamente tale comando allora risponde con un comando di ACK all'unità Master e, SOLO dopo l'invio completo dell'ultimo byte del comando di ACK, chiude il relè. A questo punto l'unità Slave avvia un timer per il controllo di timeout della chiusura del relè. Se l'unità Master non arresta il conteggio entro il termine previsto, l'unità Slave apre nuovamente il relè. Tale timer è indispensabile qualora non vi siano ulteriori unità slave che terminano correttamente la linea, compromettendo ogni comunicazione tra l'unità Master e le unità Slave registrate. La procedura continua al punto 5.

    Se l'unità Slave non risponde con un comando di ACK, l'unità Master ritenta l'invio del comando dopo un periodo di timeout. Qualora si superi il numero previsto di tentativi viene sollevata una eccezione nella procedura di registrazione (SET_ID = TRUE, SET_RELE = FALSE, RESET_TIMER_RELE = FALSE).

15. L'unità Master incrementa l'indirizzo ID_SLAVE utilizzato al passo precedente di una unità: ID_SLAVE(n+1) = ID_SLAVE(n) + 1 = n + 1.

    L'unità Master invia il comando SET_ID di assegnazione indirizzo. Tale comando ha come indirizzo del destinatario il valore ADDRESS = 0, indipendentemente dalla unità Slave a cui viene trasmesso e come unico DATA[0] = ID_SLAVE(n+1) = n + 1.

16. Se l'unità Slave riceve correttamente tale comando allora sostituisce l'indirizzo di default ID_SLAVE = 0 con quello nuovo ID_SLAVE = n + 1 e risponde con un comando di ACK all'unità Master. La procedura continua al punto 7.

    Se l'unità Slave non risponde con un comando di ACK, l'unità Master ritenta l'invio del comando dopo un periodo di timeout. Qualora si superi il numero previsto di tentativi significa che non sono presenti ulteriori unità Slave sulla linea. In tale situazione, l'unità slave di indirizzo SLAVE_ID = n, al raggiungimento del timeout di controllo chiusura relè, aprirà nuovamente il relè n-esimo, ripristinando la terminazione della linea.

17. L'unità Master invia all'unità Slave di indirizzo SLAVE_ID = n, il comando RESET_TIMER_RELE di arresto del timer di controllo del timeout del relè.

18. Se l'unità Slave di indirizzo SLAVE_ID = n riceve correttamente tale comando per l'arresto del timer di controllo del timeout del relè risponde con un comando di ACK all'unità Master. La procedura continua al punto 9.

    Se l'unità Slave di indirizzo SLAVE_ID = n non risponde con un comando di ACK, l'unità Master ritenta l'invio del comando dopo un periodo di timeout. Qualora si superi il numero previsto di tentativi viene sollevata una eccezione nella procedura di registrazione (SET_ID = TRUE, SET_RELE = TRUE, RESET_TIMER_RELE = FALSE).

19. L'unità Master invia il comando SET_RELE di chiusura relè all'unità Slave corrente di indirizzo ID_SLAVE = n + 1 per collegare fisicamente la successiva unita Slave della linea. In tale comunicazione l'indirizzo del destinatario non è più quello di default ma è il valore ADDRESS = ID_SLAVE = n + 1.
20. Se l'unità Slave riceve correttamente tale comando per la chiusura del relè, risponde con un comando di ACK all'unità Master e, SOLO dopo l'invio completo dell'ultimo byte del comando di ACK, chiude il relè. A questo punto l'unità Slave avvia un timer per il controllo di timeout della chiusura del relè. La procedura continua al punto 5. Se l'unità Slave non risponde con un comando di ACK, l'unità Master ritenta l'invio del comando dopo un periodo di timeout. Qualora si superi il numero previsto di tentativi viene sollevata una eccezione nella procedura di registrazione (SET_ID = TRUE, SET_RELE = FALSE, RESET_TIMER_RELE = FALSE).

Vengono iterati i punti dal 5. al 10.

La procedura di registrazione è terminata.

MANTENIMENTO COMUNICAZIONE

Per mantenere la comunicazione, è necessario interrogare in polling ogni dispositivo presente sul bus richiedendone lo status con un refresh rate superiore a 0,1Hz.

SEQUENZA DI DE-REGISTRAZIONE

Per poter modificare il numero di unità slave di una sottorete ovvero aggiungere o rimuovere unità SLAVE, è necessario una operazione di de-registrazione del bus.

Questa operazione si riduce alla semplice attesa del timeout di comunicazione (10 secondi), scaduto il quale si procede alla la chiusura del relè in posizione di terminazione e la reinizializzazione del dispositivo con ID_SLAVE = 0.

COMANDI DI PROTOCOLLO

RICHIESTA STATUS SLAVE

Descrizione: inoltra allo slave la richiesta di status.

Lo slave risponde con una sequenza di N byte che descrivono lo stato del dispositivo.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Richiesta Status Slave	ID	0	1	-
Invio ACK	ID	10	1	<DATA[0..10]>
Invio No ACK	ID	1	1	CODICE ERRORE

PAYLOAD STATUS

BYTE 1 - error flags:
- bit 1: reserved
- bit 2: temperature error flag
- bit 3: dsp error flag
- bit 4: generic error flag
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

BYTE 2 - mute and peak flags:
- bit 1: mute flag channel 1
- bit 2: peak flag channel 1
- bit 3: mute flag channel 2
- bit 4: peak flag channel 2
- bit 5: reserved
- bit 6: reserved
- bit 7: reserved
- bit 8: reserved

BYTE 3 - channel 1 signal value

BYTE 4 - channel 2 signal value

BYTE 5 - reserved

BYTE 6 - reserved

BYTE 7 - live flags:

- bit 1: paging on air (1 on air, 0 no active)
- bit 2: OUT A busy
- bit 3: OUT B busy
- bit 4: reserved
- bit 5: reserved
- bit 6: reserved
- bit 7: not used
- bit 8: not used

BYTE 8 - reserved

BYTE 9 - reserved

BYTE 10 - reserved

INVIO ID SLAVE

Descrizione: assegna allo slave un nuovo indirizzo (L'indirizzo di partenza è sempre 0).

Lo slave risponde con la sequenza ACK o No ACK.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Invio ID Slave	0	1	2	ID
Invio ACK	ID	0	2	-
Invio No ACK	ID	1	2	CODICE ERRORE

APERTURA RELÈ SLAVE

Descrizione: impone l'apertura del relè (posizione di bypass INPUT to LINK, questo slave non sarà più terminato).

Lo slave risponde con la sequenza ACK o No ACK prima di eseguire la commutazione.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Apertura Relè Slave	ID	0	3	-
Invio ACK	ID	0	3	-
Invio No ACK	ID	1	3	CODICE ERRORE

CHIUSURA RELÈ SLAVE

Descrizione: impone la chiusura del relè, la linea di LINK viene disattivata e lo slave termina il canale di comunicazione.

Lo slave risponde con la sequenza ACK o No ACK prima di eseguire la commutazione.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Chiusura Relè Slave	ID	0	4	-
Invio ACK	ID	0	4	-
Invio No ACK	ID	1	4	CODICE ERRORE

ARRESTO TIMER RELÈ

Descrizione: ferma il countdown fissato in 5 secondi al termine del quale verrebbe imposta la terminazione sullo slave (chiusura relè).

Lo slave risponde con la sequenza ACK o No ACK.

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
Arresto Timer Relè	ID	0	5	-
Invio ACK	ID	0	5	-
Invio No ACK	ID	1	5	CODICE ERRORE

COMANDI DMA 82 / DMA 162

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
COMANDO	ID	0	CODE	-
Invio ACK	ID	N	CODE	<DATA[0..10]>
Invio No ACK	ID	1	CODE	CODICE ERRORE

PAYLOAD STATUS

INIT COMMANDS

CODE 0x11 **INIT GENERAL**

BYTE 1 - revisione fw

BYTE 2 - general flags:

- bit 1: stereo/mono flag OUT A e B
- bit 2: reserved
- bit 3: bridge mode flag OUT A e B
- bit 4: reserved
- bit 5: reserved
- bit 6: reserved
- bit 7: reserved
- bit 8: reserved

BYTE 3 to 8 - reserved

CODE 0x12 **INIT OUTPUT**

BYTE 1 to 4 - volume OUT A

BYTE 5 - OUT A channel flags

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used

- bit 8: not used

BYTE 6 to 8 - reserved

BYTE 9 - input code OUT A

BYTE 10 to 31 - reserved

BYTE 32 to 35 - volume OUT B

BYTE 36 - OUT B flags:

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

BYTE 37 to 39 - reserved

BYTE 40 - input code OUT B

BYTE 41 to 124 - reserved

CODE 0x13 INIT INPUT

BYTE 1 to 4 - stereo volume IN1

BYTE 5 to 8 - mono volume IN1 L

BYTE 9 to 12 - mono volume IN1 R

BYTE 13 to 48 - reserved

BYTE 49 to 52 - stereo volume IN2

BYTE 53 to 56 - mono volume IN2 L

BYTE 57 to 60 - mono volume IN2 R

BYTE 61 to 96 - reserved

BYTE 97 to 100 - stereo volume EXT

BYTE 101 to 104 - mono volume EXT L

BYTE 105 to 108 - mono volume EXT R

BYTE 109 to 144 - reserved

BYTE 145 to 148 - mono volume BAL IN

BYTE 149 to 160 - reserved

BYTE 161 to 164 - mono volume PAGING

BYTE 165 to 176 - reserved

OUTPUTS COMMANDS

CODE 0x31 SET VOLUME OUTPUT

BYTE 1 - codice output

BYTE 2 to 5 - valore guadagno

CODE 0x32 SET GENERAL VOLUME OUTPUT

BYTE 1 to 4 - valore guadagno OUT A

BYTE 5 to 8 - valore guadagno OUT B

BYTE 9 to 12 - valore guadagno OUT C

BYTE 13 to 16 - valore guadagno OUT D

CODE 0x35 ROUTING

BYTE 1 - codice output

BYTE 2 - codice input

CODE 0x3A SET MUTE OUTPUT

BYTE 1 - codice output

BYTE 2 - mute flag:

- bit 1: mute/unmute flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODE 0x3B SET MUTE GENERALE OUTPUT

BYTE 1 - mute flag:

- bit 1: mute/unmute flag OUT A (1/0)
- bit 2: mute/unmute flag OUT B (1/0)
- bit 3: mute/unmute flag OUT C (1/0)
- bit 4: mute/unmute flag OUT D (1/0)
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODE 0x3E **GENERAL ROUTING**

BYTE 1 - codice input

CODE 0x3F **SET VOLUME STEREO/BRIDGE OUTPUT**

BYTE 1 - codice output (1 per coppia OUT A-B)

BYTE 2 to 5 - valore guadagno uscita OUT A

BYTE 6 to 9 - valore guadagno uscita OUT B

CODE 0x40 **SET MUTE STEREO/BRIDGE OUTPUT**

BYTE 1 - codice output (1 per coppia OUT A-B)

BYTE 2 - mute flag:

- bit 1: mute/unmute flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODE 0x41 **ROUTING STEREO**

BYTE 1 - codice canali output (1 per coppia OUT A-B)

BYTE 2 - codice input

CODE 0x42 **GENERAL VOLUME INPUT**

BYTE 1 to 4	- valore guadagno IN1 L
BYTE 5 to 8	- valore guadagno IN1 R
BYTE 9 to 12	- valore guadagno IN2 L
BYTE 13 to 16	- valore guadagno IN2 R
BYTE 17 to 20	- valore guadagno EXT L
BYTE 21 to 24	- valore guadagno EXT R
BYTE 25 to 28	- valore guadagno BAL IN
BYTE 29 to 32	- valore guadagno PAGING

INPUTS COMMANDS

CODE 0x51 **SET VOLUME INPUT MONO**

BYTE 1	- codice input
BYTE 2 to 5	- valore Guadagno

CODICI OUPUT

OUT A	0x01
OUT B	0x02

CODICI INPUT

IN1 L+R	0x01
IN2 L+R	0x02
EXT L+R	0x03
BAL IN	0x04
PAGING	0x05
IN1 L	0x06
IN1 R	0x07
IN2 L	0x08
IN2 R	0x09
EXT L	0x0A
EXT L	0x0B

COMANDI DMA 162P

<FUNCTION>	<ADDRESS>	<LENGTH>	<CMD>	<DATA[0..N-1]>
COMANDO	ID	0	CODE	-
Invio ACK	ID	N	CODE	<DATA[0..10]>
Invio No ACK	ID	1	CODE	CODICE ERRORE

PAYLOAD STATUS

INIT COMMANDS

CODE 0x11 **INIT GENERAL**

BYTE 1 - revisione fw

BYTE 2 - general flags:

- bit 1: stereo/mono flag OUT A e B
- bit 2: reserved
- bit 3: bridge mode flag OUT A e B
- bit 4: reserved
- bit 5: reserved
- bit 6: reserved
- bit 7: reserved
- bit 8: reserved

BYTE 3 to 8 - reserved

CODE 0x12 **INIT OUTPUT**

BYTE 1 to 4 - volume OUT A

BYTE 5 - OUT A channel flags

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used

- bit 8: not used

BYTE 6 to 8 - reserved

BYTE 9 - input code OUT A

BYTE 10 to 31 - reserved

BYTE 32 to 35 - volume OUT B

BYTE 36 - OUT B flags:

- bit 1: reserved
- bit 2: reserved
- bit 3: mute flag
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

BYTE 37 to 39 - reserved

BYTE 40 - input code OUT B

BYTE 41 to 124 - reserved

CODE 0x15 INIT INPUT

BYTE 1 to 4 - stereo volume IN1

BYTE 5 to 8 - mono volume IN1 L

BYTE 9 to 12 - mono volume IN1 R

BYTE 13 to 48 - reserved

BYTE 49 to 52 - mono volume CH1

BYTE 53 to 64 - reserved

BYTE 65 to 68 - mono volume CH2

BYTE 69 to 80 - reserved

BYTE 81 to 84 - mono volume CH3

BYTE 85 to 96 - reserved

BYTE 97 to 100 - mono volume CH4

BYTE 101 to 112 - reserved

OUTPUTS COMMANDS

CODE 0x31 **SET VOLUME OUTPUT**

BYTE 1 - codice output

BYTE 2 to 5 - valore guadagno

CODE 0x35 **ROUTING**

BYTE 1 - codice output

BYTE 2 - codice input

CODE 0x3A **SET MUTE OUTPUT**

BYTE 1 - codice output

BYTE 2 - mute flag:

- bit 1: mute/unmute flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used
- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

INPUTS COMMANDS

CODE 0x51 **SET VOLUME INPUT MONO**

BYTE 1 - codice input

BYTE 2 to 5 - valore guadagno

GENERAL COMMANDS

CODE 0x26 **SET BUS LOCAL MODE**

BYTE 1 - BUS/LOCAL

- bit 1: local/bus mode flag (1/0)
- bit 2: not used
- bit 3: not used
- bit 4: not used

- bit 5: not used
- bit 6: not used
- bit 7: not used
- bit 8: not used

CODICI OUPUT

OUT A	0x01
OUT B	0x02

CODICI INPUT

CH1	0x01
CH2	0x02
CH3_IN1_R (*)	0x03
CH4_IN1_L (*)	0x04
IN1 L+R	0x05

(*) Il canale bus 3 e 4 sono fisicamente sovrapposti con gli ingressi RCA1L e RCA1R. Se si vuole interagire con uno di questi ingressi, accertarsi di essere nella corretta modalità, locale per RCA o bus per i channel.

Note

1. Quando l'amplificatore è in modalità locale e in stereo mode, per modificare il volume/mettere in mute i canali di uscita inviare il comando Set Volume Output/Set Mute Output sia al canale A che al canale B.
2. Quando l'amplificatore è in modalità locale o bus e in bridge mode, per modificare il volume/metter in mute i canali di uscita inviare il comando Set Volume Output/Set Mute Output solo al canale A.

RCF S.p.A. Via Raffaello Sanzio, 13 - 42124 Reggio Emilia – Italy

Tel +39 0522 274 411 - Fax +39 0522 232 428 - e-mail: info@rcf.it - **www.rcf.it**