SL-83 Programmable access controller / event logger

Technical description

Version: PCB:R7A FW:8303x

Supersedes: 8302x

1. Advance notes

1.1. ID media, reader modules

The following text assumes that the SL-83 controller is used with iButtons as the ID medium and simple, passive, contact boxes for reading iButtons. There are other reader modules that can be connected to the SL-83 instead of the iButton reader boxes. These include readers for different ID media (magnetic cards, RF cards and tags, etc.) and extended functionality modules such as readers with PIN keyboards. This technical description will, however, assume that the simple iButton boxes are used (except where specifics of the various modules are discussed). This is for the sake of clarity and helps keep the manual slightly easier to digest.

1.2. Communication

The text assumes the basic method of communication between the controller and the master PC, i.e. the current loop, which connects to the PC's COM port. The SL-83R7 controller has an onboard Tibbo NetModule, enabling direct connection to a network.

2. General system description

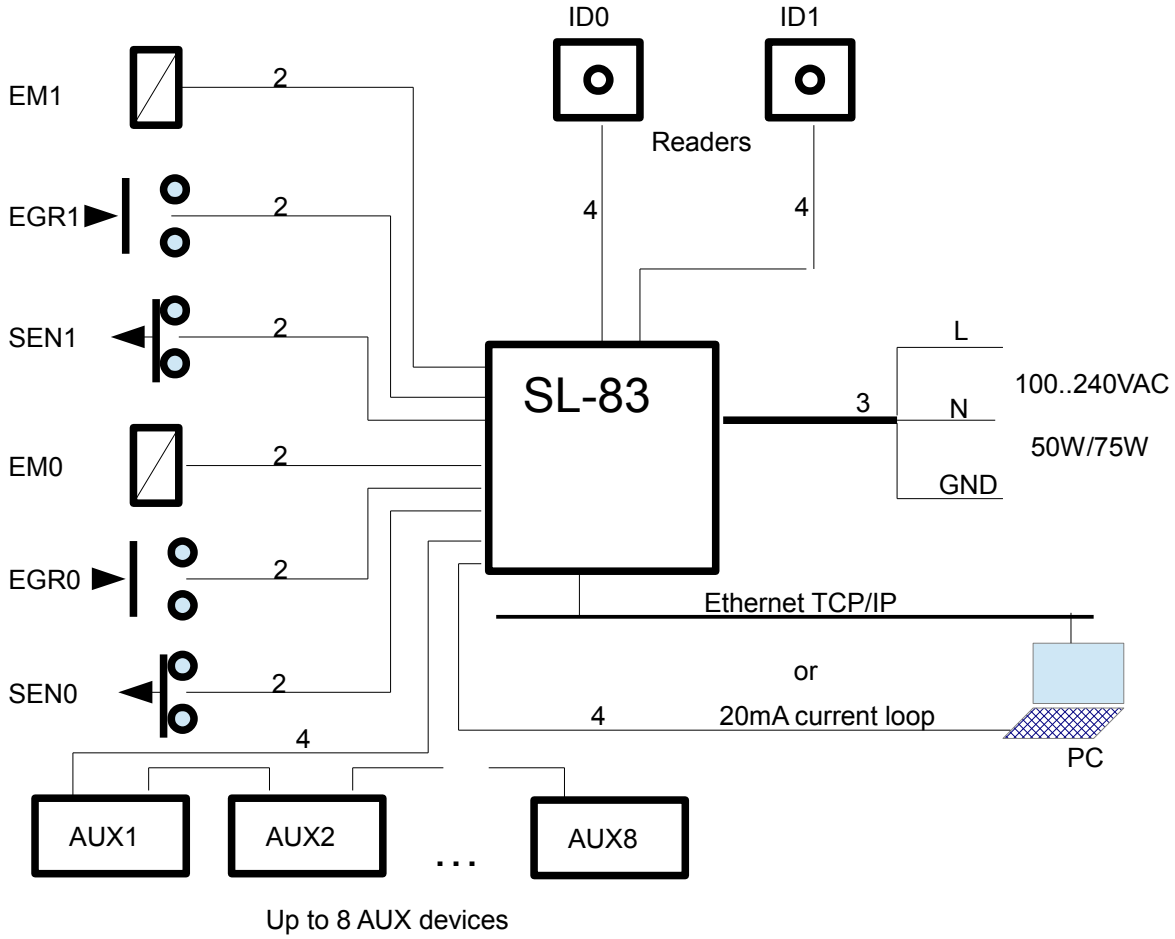


Fig1.

EM0,EM1	actuators: door strikes, turnstiles (settable to 12V or potential-free relay output)
EGR0,EGR1	potential-free, normally open inputs, primarily for egress pushbuttons
SEN0,SEN1	potential-free, normally closed inputs, primarily for door sensors (reed or microswitch)
AUX1..AUX8	various additional modules attachable in a chain, e.g. readers, terminals, clock displays, etc.
ID0, ID1	reader modules for iButtons or RF tags

The SL-83 system is used for controlling the access of people to a certain room or building and/or logging their arrival and departure times for the purpose of security or time & attendance reporting.

An event record is the act of logging the event type, the current time and date and the code of the iButton or RF tag (if it's an ID event). This data is transferred to a PC at a convenient time for further processing.

ID events i.e. events that include a person's ID badge code are also called clockings. Other events can also be logged, such as the opening and closing of doors (SEN0,SEN1), the activation of egress buttons (EGR0,EGR1) and even the state of the AC input.

Access control is achieved by an electric door strike, turnstile or any other type of actuator. The most common is an electric door strike that is installed in the door frame. The lock can still be opened the old fashioned way with a key in case of system malfunction or a lengthy power cut. In normal circumstances the door is opened by making a clocking with an iButton or RF tag.

Two readers can be connected to a controller unit (although more can be attached via AUX devices). Readers have a contact cup for iButtons or an antenna for RF tags and two coloured LEDs to indicate a successful read and the activation of the corresponding relay output.

The controller unit is in a metal cabinet (32x37x8 cm). The same cabinet houses an AC/DC converter that supplies power to the controller board and a 12V 7Ah sealed lead-acid battery that provides a certain length of independent operation.

The controller unit is connected to a master PC via a serial link or directly to an ethernet TCP/IP network.

The serial link is an isolated 20mA current loop and the distance to the PC can be up to 1000m.

The PC performs the functions of acquiring the records, storing them on disk and generating reports. It also serves the purpose of generating access tables for the employees and uploading them to the controller. The tables determine the conditions under which a certain iButton is 'active' i.e. can open the door. After these tables are uploaded to the controller it functions autonomously, making access-rights decisions without communicating with the PC. The PC is also used to configure the controller, which is done each time the basic functions of the terminal are changed. The configuration is held in the serial EEPROM of the unit and tells the controller how to react to the events that are generated i.e. which events make records and which relays (if any) are activated.

The PC performs the following functions:

- downloading and processing of clocking data (KatzeReports.EXE, Comm8x.EXE)
- entry, editing and uploading of the access rights table (Kata.EXE, Tab8x.EXE)
- configuration of the controller (Cnf8303.EXE)
- uses the data stored by Comm8x for generating views and reports on employee presence, working hours, etc.

2.1. Hardware

Comprises:

- controller PCB SL83R7A
- power supply
- metal case with hinged door (houses PCB, supply and battery)
- Readers: contact boxes for iButtons, RF tag readers etc.
- AUX devices in a daisy-chain
- ID badges: DS1990A iButtons, RF tags, RF cards, etc.
- SL-253 RS-232/Current loop converter (in case of connection to a PCs serial port)

2.2. Controller PCB

- 128kB battery-backed SRAM
- Real Time Clock chip
- 256x16 serial EEPROM for configuration data
- 2 relay outputs, all 1.5A max - resettable fuse protected, individually configurable to:
  - 12V/NO 12V output, normally open (default setup, used for most door strikes)
  - 12V/NC 12V output, normally closed (used for fail-safe latches i.e. latches that are locked while powered
  - NV/NO No Voltage, normally open (30V max) -if an external source is to be used for the actuator
  - NV/NC No Voltage, normally closed (30V max) - if an external source is to be used for the actuator
- 2 sensor inputs (for reed or microswitch door sensors)
- 2 egress button inputs
- LCD display: alphanumeric with backlight, 24x2 characters

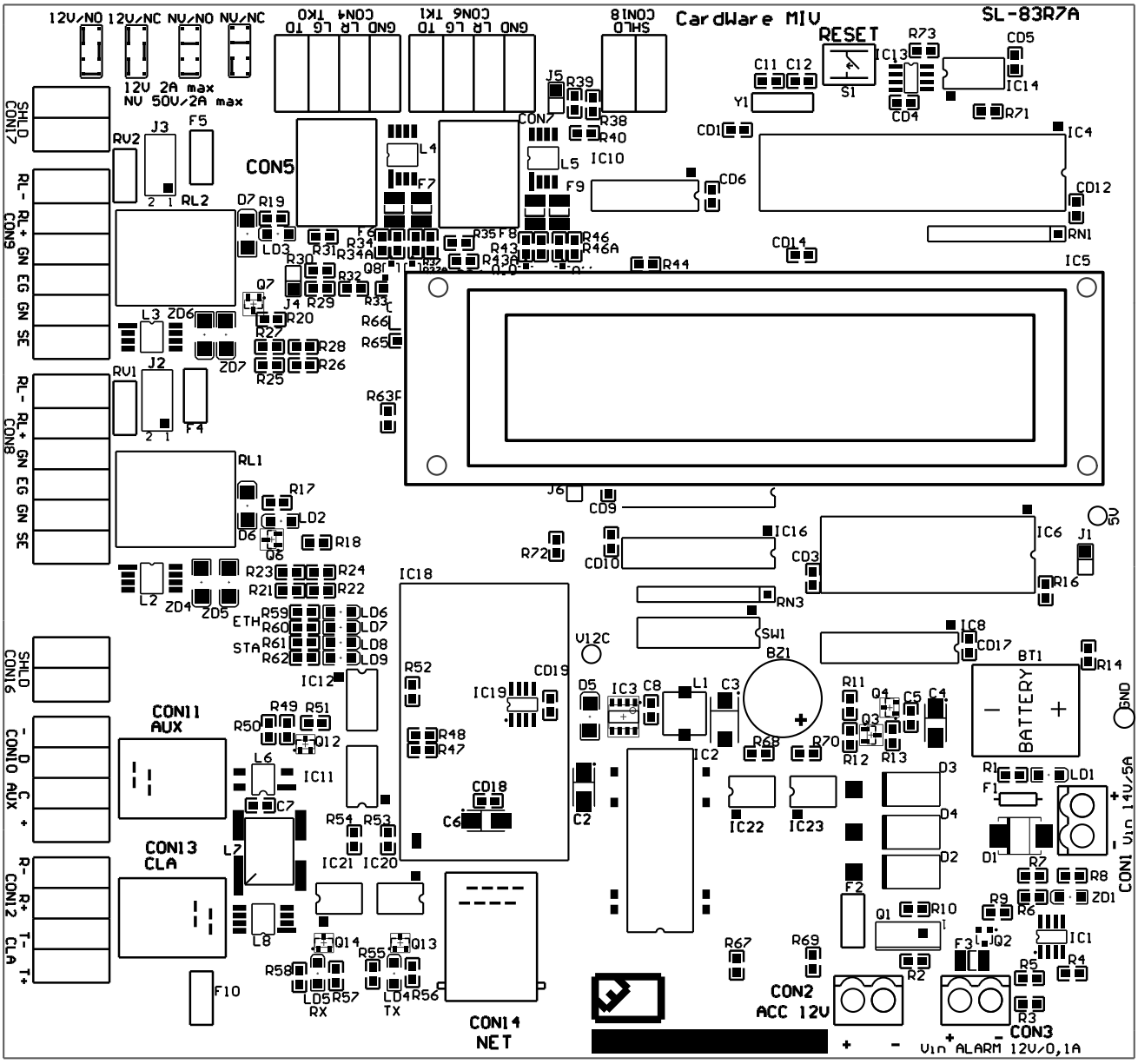


Fig.2

✶ DIP switch (language, controller ID code)

The controller ID is in the range (00H – 3FH). This code is sent as the last character of each record. The purpose of this is to know the source controller for each record in the case that all data is saved in the same file. If there is more than one controller in the system, the DIP switches should be set so that the addresses differ.

The address is set with switches SW1 to SW6 of the dip switch. SW1 is the lowest and SW6 is the highest bit of the address. If a switch is in the ON position, the respective bit is a ‘0’, if it is in the OFF position it is a ‘1’.

The ID code of the controller is also used by the report generation program. The two readers of each controller generate clockings with the event codes 0 and 1 respectively. These codes are transformed into events such as ENTER, EXIT, LUNCH BREAK etc. according to a table that can be freely configured. (This is done by the KatzeReports program.)
If there are several controllers in a system, each one can have a different association of readers to events; the report generation program decides which event to associate to a clocking based on the event code and ID of the controller.

Switches SW7 and SW8 set the language that is used for the service messages on the LCD.

SW7	SW8	language
ON	ON	Hungarian
ON	OFF	English
OFF	ON	Serbian
OFF	OFF	Greek

2.3. Power Supply

1.) Connection plate with terminal blocks and fuse

The connection plate is where the 220VAC is connected by a detachable terminal block. Care must be taken so that the live wire goes into the block marked (L). This is to ensure that the live wire goes through the fuse on the connection plate. Ground earth must be used and it is the installer's responsibility to check its availability and quality.

There is also a fuse on the LIVE input and varistors across the LIVE-NEUTRAL and LIVE-EARTH lines.

2.) Ferrite ring around 220V supply cable

This is for suppressing high frequency symmetrical (common-mode) interference. In certain noisy environments it may be necessary to add a few more ferrite rings.

3.) AC/DC converter

This converts 100-240VAC into 14VDC. Depending on versions, the power rating of the converter can be 50W or 75W. The nominal output voltage of the AC/DC converter is 15V, but it is set to 14V by a small variable resistor next to the terminal contacts.

4.) Sealed 12V/7Ah lead-acid battery

The battery provides autonomy in case of a power cut. Calculating the approximate duration of autonomy is only possible if the battery is in good condition and fully charged. We can only estimate the static power consumption and can only guess at variable drain on the battery e.g. high current door strike solenoids that draw power only when active. This depends on the number of door openings and the relay time and whether the door sensor input is used to power off the relay output when the door is closed again.

Approximate values:

12V 7Ah battery = 84Wh

SL-83R7 board with no relays active: cca. 3W

RF reader: cca 3W i.e two RF readers = 6W

AUX devices: add power of all attached AUX devices (the max. drain on the AUX channel is around 15W)

Charging

The battery is trickle-charged with a constant voltage of around 13.7V and a maximum current of 2.5A.

Discharge protection

If the battery voltage falls below 10V, the controller is switched off and the drain of the battery becomes just a few mA. Even so, always disconnect the battery if the controller is disconnected from the AC supply for a longer period.

2.4. Contact box (iButton reader)

This is an aluminium box with mounting holes for screws on the back plate. It holds the iButton contact cup and two coloured LEDs.
Signalling:

- read OK -green or yellow
- relay activated -red

It should be noted here that the red led always lights up on contact box n (n=1,2) when relay n is activated.

The activation of relay n can happen for various reasons, depending on the configuration of the contreoller. It can happen, for instance, due to the touching of an iButton to a *different* contact box or the closing of an egress input. As mentioned, there are alternative readers, but these will be dealt with in a later section. Also, see the technical descriptions of the respective readers.

2.5. Connectors and cables

Input/Output 6-pin screw terminals (CON8, CON9)

The screw terminals on the top left of the controller board provide access to the I/O functions of the 2 channels.

These are:

1. Sensor input
2. Egress input
3. Actuator (relay) output: 12V/2,5A or voltage-free

Fig.3 is a diagram of one channel (one of the two 6-pin screw terminals):

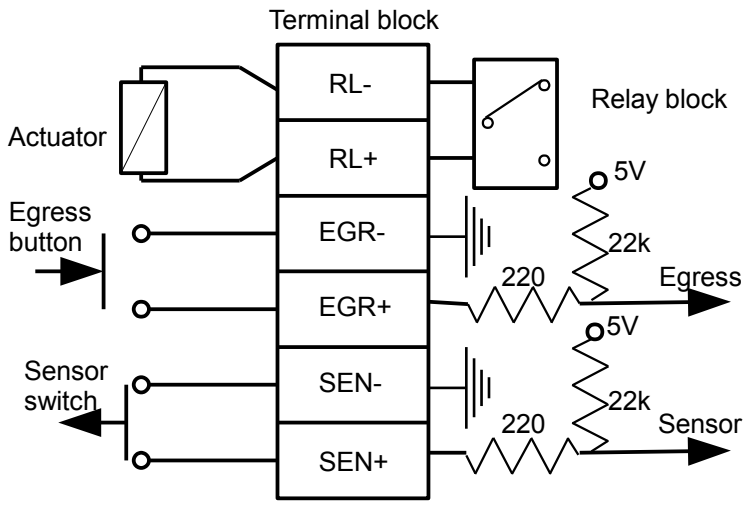


Fig.3

Notes:

- the Egress and Sensor inputs have additional EMC and overvoltage protection that is not shown on this diagram
- The relay output terminals are connected with a 30V varistor, so no voltages higher than 28V should be used

The two relay blocks can each be configured using a set jumpers:

- **12V/NO** 12V Normally Open
  Default: no voltage
  12V appear across the RL+,RL- terminals when the relay is active
  Use for: 12V door strikes
- **12V/NC** 12V Normally Closed
  Default: 12V
  No voltage across the RL+,RL- terminals when the relay is active
  Use for: fail-safe (fire) door strikes, electromagnetic locks
- **NV/NO** No Voltage Normally Open
  Default: relay open

Use for: externally powered (e.g. 24V) actuators

- **NV/NC** No Voltage Normally Closed
  Default: relay closed
  Use for: externally powered (e.g. 24V) actuators

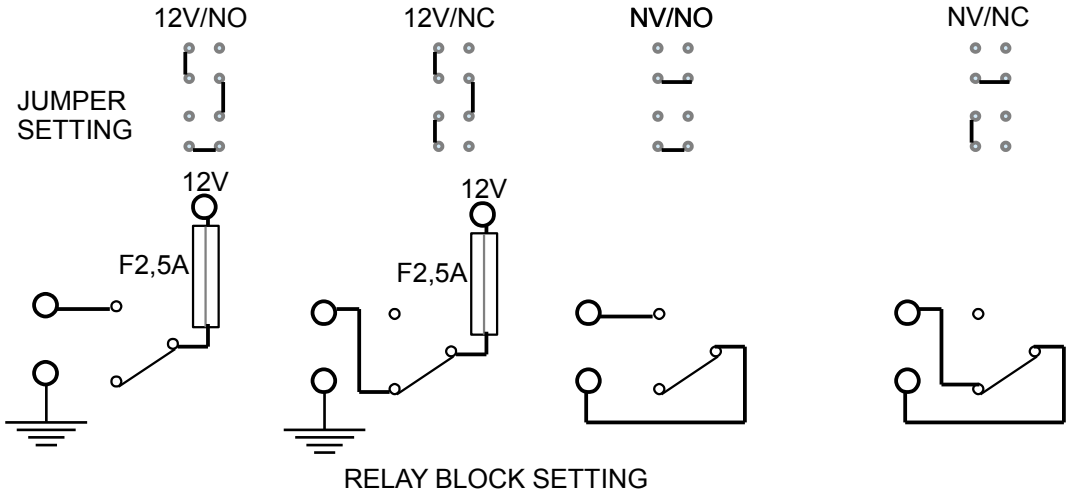


Fig. 4 Jumper settings for relay blocks

Fig. 4 shows the four possible relay block configurations and their corresponding jumper settings.

Readers (TK0, TK1)

The readers have both a telephone (RJ-11) connector and a detachable 4-pin terminal block. These connectors are connected in parallel, so either can be used, depending on the type of cable used.
Cable length should not exceed 20m.

RJ-11 conntectors (CON5, CON7)

Flat 4-wire telephone cables can be used, with 6/4 phone plugs (RJ-11).The cable used for the readers is such that the little plastic levers of the plugs should be on the same side of the cable (either the side with the stripe or the one without) on both ends of the cable.

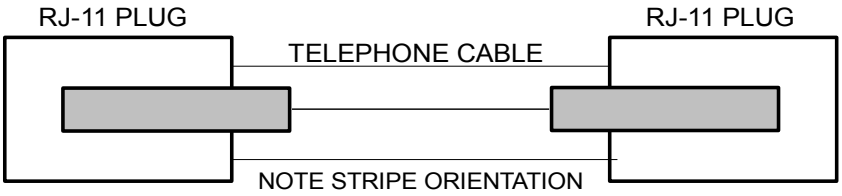


Fig. 5: flat phone cable with RJ-11 plugs

4-pin screw terminals (CON4, CON6)

Some reader modules do not have screw terminals, just RJ-11 sockets. In this case an RJ-11/screw terminal adapter must be used. Markings on the reader's screw terminal or the adapter are either the same as on CON4, CON6 or are shown in the table below.
Wires should be connected one-to-one i.e. GND to GND, LR to LR(L1) etc.

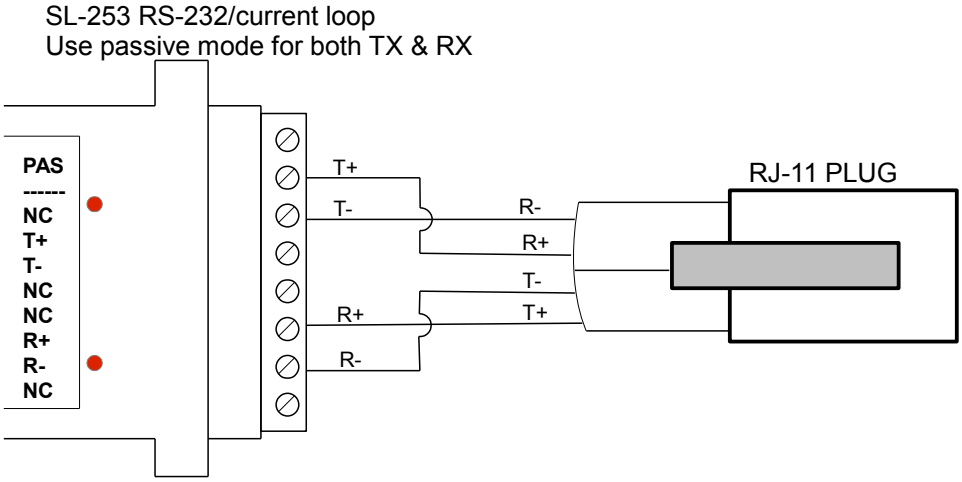
CON4/CON6	Reader/Adapter
GND	GND
LR	L1
LG	L2
TD	D

Serial communication - Current Loop Active (CON12, CON13)

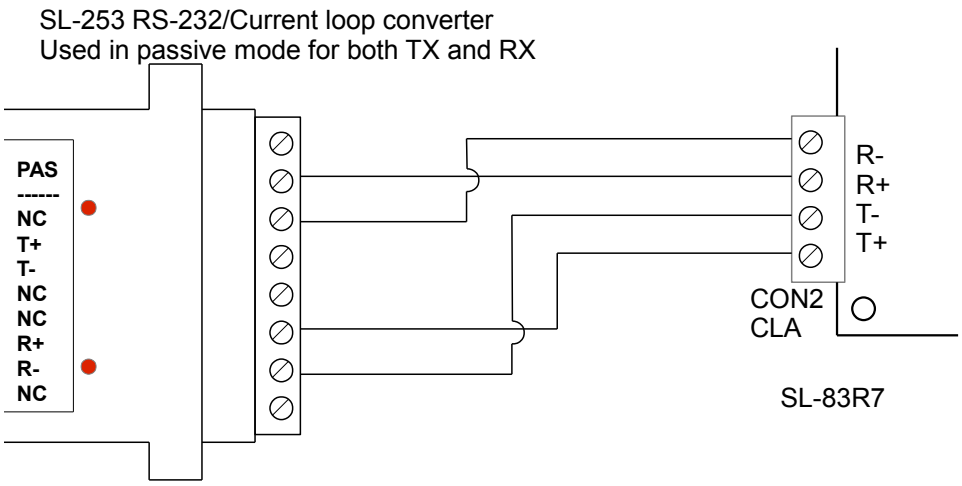
The SL-83R7 board has an active current loop channel, with two connectors: CON12 - 4 pin detachable screw terminal and CON13 - top-entry 6/4 RJ-11 socket.
Since the current loop interface is active, meaning that the 20mA current is supplied for both the RX and TX channels by the SL-83R7 board, the SL-253 current loop converter should be wired as "passive".
Cable length can be up to 1000m.

Note that there is no jumper to select the type of communication used, serial or ethernet. Serial communication over the current loop can occur even if the Tibbo Ethernet module is installed, provided that the two are not used at the same time.

RJ-11 connector (CON13)



4-pin terminal block (CON12)



Input voltage - 2 pin terminal block (CON1)

CON1 is the input connector for the 14VDC the SL-83 uses for normal operation.

The input is protected against reverse polarity and high input voltages by a 5A/30V self-healing fuse and a 15,3V/5A zener diode.

The presence and correct polarity of the input voltage is shown by a red LED (LD1) just above CON1.

Battery connector - 2 pin terminal block (CON2)

CON2 is used to connect a sealed lead-acid battery (12V/7Ah) to the SL-83. The battery is charged through a self healing 2,5A/30V fuse that acts as a current limiter in case the battery is faulty or totally discharged. The open-source voltage (i.e. the trickle charge volatge) on CON1 should be around 13,7-13,8V.

Input voltage low ALARM - 2 pin terminal block (CON3)

CON3 is the 12V/100mA alarm output for signalling a missing or low voltage on CON1 (input voltage).

If the input voltage falls below 12,3V, a voltage of 12V appears on CON3. This can be used for a or a signal lamp.

The output is protected by a self-healing 140mA fuse, so consumption should be considered when choosing the signalling method.

Shield - 2 pin terminal block (CON6, CON7, CON8)

CON6, CON7 and CON8 are connected to the PCBs ground plane and the metal case. In case of EMI problems or just for the sake of proper engineering practice, the shields of used shielded cables can be connected to these connectors.

2.6. Firmware 8303x (x is a version letter a,b,…)

The firmware has the following functions:

- After a reset, self-test routines are called. The LCD is initialised and a non-destructive test of the internal and external RAM is made. The ROM checksum is also checked.
  During the self-test procedure, the following screen is shown (just before the end of the self-test routine):

F	**D**	**3**	**1**		**D**	**:**	**7**	**1**		**X**	**0**	**1**	**F**	**F**	**I**	**C**							
A	**S**	**M**	**:**	**2**	**.**	**0**	**a**		**C**	**:**		**8**	**3**	**0**	**3**	**i**							

1. The 4-digit hex number in the top-left corner is a reset-code, which gives an indication of where the program was when the reset occurred. Since a reset can happen due to a watchdog timeout, this code is very useful in tracking down possible firmware bugs.
2. The byte after ‘D;’ is the value set on the DIP switch
3. X represents the start of the XRAM test, the four hex characters represent the page counter
4. I represents the start of the internal RAM test
5. C represents the start of the ROM checksum test
6. the lower row provides information on the versions of the assembler and C parts of the firmware

After the self-test routines, the integrity of the buffered clocking data and pointers is checked. If the pointers have been corrupted, the controller resets them. This will be indicated by a ‘0’ registration counter i.e. an empty buffer.

If the buffer pointers are corrupt after a reset, the FATAL_RESET COUNTER is incremented, otherwise the RESET COUNTER is incremented. These two counters can be read out with a Status command. They are revolving 16-bit counters and are useful in tracking down problems like missing clocking data or corrupted access tables.

If these counters show a tendency to increment, that means that there is most probably an EMI problem, and additional filtering is needed for the power lines (off-line UPS, isolation-transformer, additional ferrites, etc.).

The problem may also be caused by bad earthing i.e. the protective ground wire is not connected to a proper low-resistance ground.

- scans the sensor and egress inputs and executes the actions configured in the **ActionWord** corresponding to the occurred event
- reads the TK0, TK1 reader inputs and reads badges (iButtons/RF tags) and controls the signal LEDs
- if so configured and if the random event conditions are met, executes the ActionWord associated to that reader input (spot searches, alcohol tests etc.)
- Looks for the code of the badge in the ID table, gets the **ActionByte** and makes a clocking record: activity (number of the reader where the badge was read), time & date, badge code.
- Using the **ActionByte** the controller decides on further action (see. Tables)
- The sensor inputs are meant for monitoring whether a door is open or closed, but can be used for monitoring the state of other voltage-free contacts. Each sensor input can generate two events: one when the input is shorted, the other when it is opened.
- The egress inputs are meant to be used with push-to-make buttons. They can be used to activate relay outputs or to monitor the state of any voltage-free contacts. Each egress input can generate two events: one when the input is shorted, the other when it is opened.
- From time to time the controller tries to transmit the accumulated data (records) to the master PC
- Receives and recognises direct commands from the PC and carries them out.
- Communicates with up to 8 AUX devices, sending and receiving data

- AUX based ID readers and terminals can send ID codes which can be used to execute **ActionWords** depending on the ID tables and A, B & C bits just like when these codes are read from the TK0, TK1 inputs

The following two flowcharts illustrate how the controller treats ID events i.e. reading of any ID badge. There are also events that are generated by changes in the state of the two sensors and egress inputs. These events are not illustrated here with flowcharts, since they are much simpler; when one of these events occurs, the corresponding ActionWord is executed.

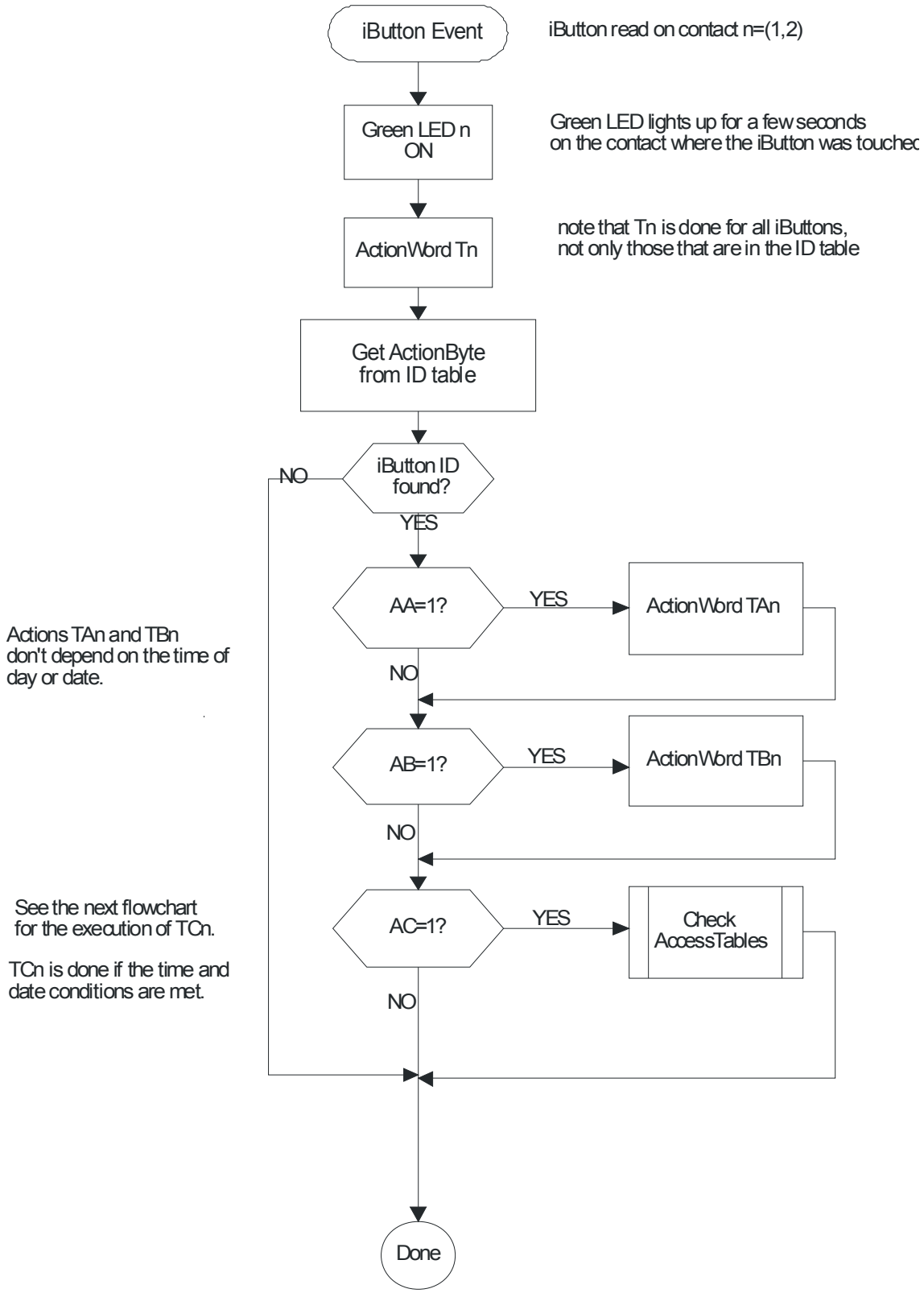


Fig. 5

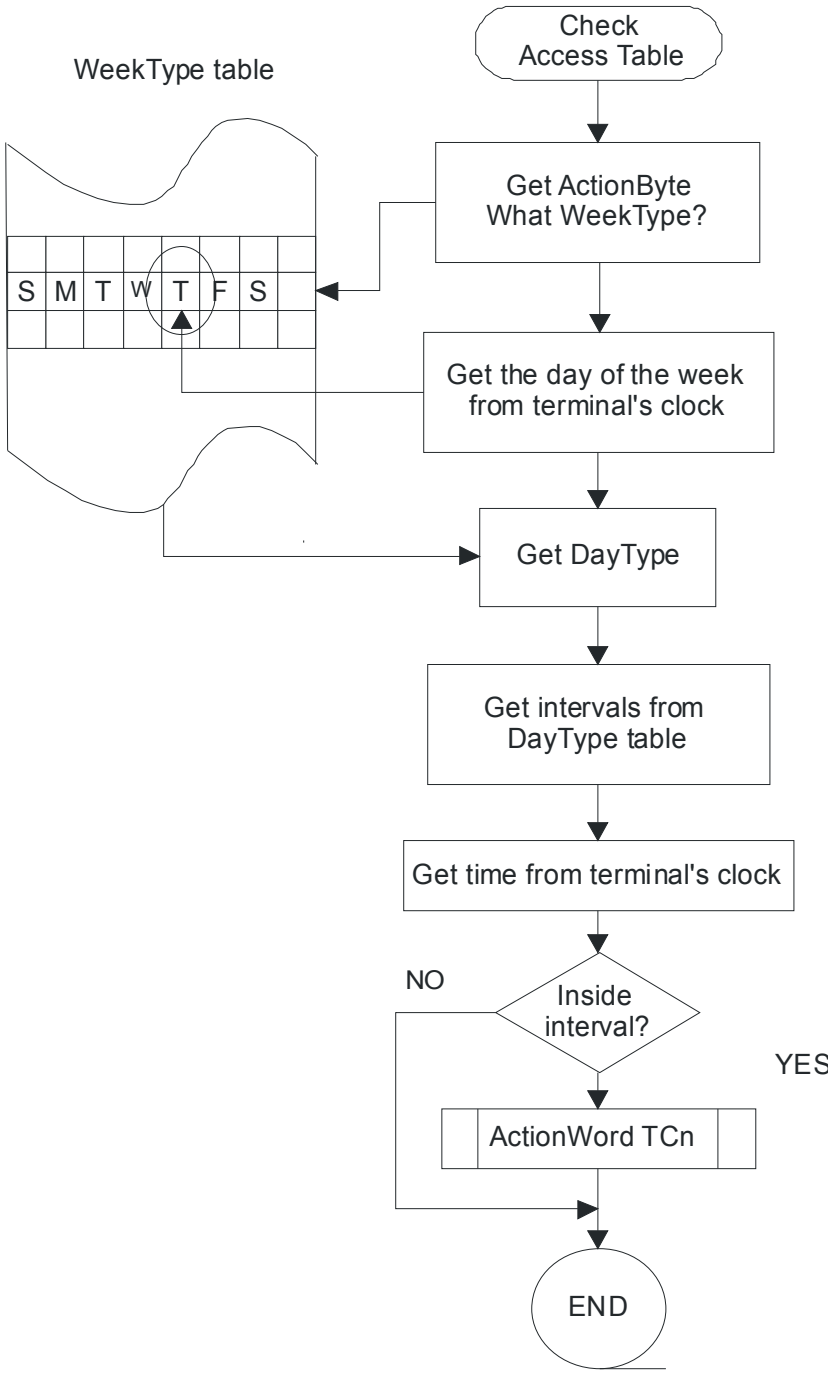

WeekType table
S M T W T F S
Check
Access Table
Get ActionByte
What WeekType?
Get the day of the week
from terminal's clock
Get DayType
Get intervals from
DayType table
Get time from terminal's clock
NO
Inside
interval?
YES
ActionWord TCn
END

Fig. 6

3. Access Tables

A certain section of the controller's non-volatile RAM memory is devoted to storing the access tables. These tables tell the controller what to do after reading a badge.

Version 8303x of the firmware uses the following tables:

- **ID table:** ID codes of badges, each with its **ActionByte**
- **DayType table:** each entry holds two time zones during the 24 hours of a day, when the relay is activated due to the reading of a badge that has this DayType associated with it on the current day of the week (Monday, Tuesday, etc.) There are 32 different DayTypes
- **WeekType:** Holds seven indexes (one for each day of the week) to DayTypes. There are 32 WeekTypes.

The main purpose of these tables is to define a "calendar" for activating the relay, which opens the door, turnstile or other access mechanism. The current version of firmware (8303x) uses these two tables in the following fashion:

- When a badge is read, the controller reads the appropriate **ActionWord** (Tn, n=1,2) from the EEPROM and proceeds accordingly (makes clockings, activates relays,..)
- it then searches the ID table for the badge code
- when it finds the table entry it gets the action byte of the badge in question
- if the AA bit is set, the TAn (n=1,2) ActionWord is executed
- if the AB bit is set, the TBn (n=1,2) ActionWord is executed
- If the AC bit is set, the controller continues checking the time-dependent access rights:
- The action byte holds a pointer to the WeekType associated to the badge
- the controller uses the current day of the week as an index into the WeekType table to get the DayType
- the current time from the RTC chip is checked against the two zones defined in the DayType entry
- if the time is within one of the two zones, the **ActionWord** for that event is retrieved and appropriate action is taken (ActionWord TCn)

(See the preceding two flowcharts for a graphical representation of this algorithm.)

3.1. ID TABLE

This table contains elements of 8 bytes. Each element comprises of 7 of the 8 bytes of a badge (iButton, RF tag) code. Instead of the 8th byte, which would be the CRC, is the ActionByte. That byte is defined by the system administrator when setting up the access rules for the system.

Action Byte

b7:	AC bit	(set if the time dependent action TC should be executed for this badge)
b6:	AB bit	(set if action TB should be executed for this badge)
b5:	AA bit	(set if action TA should be executed for this badge)
b4: -- b0:	WeekType (0 to 31)	

One, two or all three of the action bits (AA, AB and AC) can be set in a touch memory's ActionByte indicating which of three possible actions are taken when the badge is read. The AA and AB bits refer to actions that don't depend on time, whereas if the AC bit is set, the WeekType is checked and the action is executed if the date and time conditions are met.

3.2. DayType table

Holds 32 elements of 8 bytes each.
Each day represents two time zones within the 24 hours in a day when touching an active badge results in the execution of a configured TC action. These zones are defined in the following format:

FH1 FM1 TH1 TM1 FH2 FM2 TH2 TM2

FH1: From Hours 1	TH1: To Hours 1	
FM1: From Minutes 1	TM1: To Minutes 1	same for zone 2

Example:

08 30 14 00 16 00 19 50

The ID codes for which this DayType is currently active can open the door between 08:30 and 14:00 and between 16:00 and 19:50.
If we want to define a DayType for unlimited access we can set the first zone from 00:00 to 23:59. The second zone doesn't matter.

3.3. WeekType table

This table has 32 elements of 8 bytes each.
It holds pointers into the DayTypes table, one for each day of the week.

Byte 0:	DT0	day type for Sunday
Byte 1:	DT1	day type for Monday
Byte 2:	DT2	day type for Tuesday
Byte 3:	DT3	day type for Wednesday
Byte 4:	DT4	day type for Thursday
Byte 5:	DT5	day type for Friday
Byte 6:	DT6	day type for Saturday
Byte 7:	NU	not used

4. Protocol

4.1. Downloading clocking data

The controller tries to transmit clocking data to the PC by its own accord. The format and protocol depend on the mode the controller is in:

ON-LINE: the acquisition program is active on the master PC. Every clocking record is transmitted to the PC immediately. The clocking buffer is empty.

OFF-LINE: the master PC is switched off or the acquisition program is inactive. In this case the records are put into the clocking buffer (8000 records max.).

4.1.1 ON-LINE mode

SL-83⇨PC

<SOH> <'S'> <STX> (STRING) <CR> <ETX> <LRC>

PC⇨SL-83

<ACK> if transmission successful

<NAK> if transmission error: format or LRC error

Notes:

- (STRING) has the following format:
  <D10><D1><M10><M1><Y1><H10><H1><m10><m1><A><B3><B2><B1><B0><TID>
- If the controller doesn't receive an ACK in the defined timeout period, or if it receives any other character the transmission is unsuccessful, no further transmission is attempted and the record is stored into the controller's buffer.
- <D10> to <m1> are the characters of the time/date field
- <A> is the event code (Event codes are given in the table in section .)
- B3 to B0 are the 4 least significant bytes of the unique badge code
- TID is the controller ID code set by the DIP switch on the SL-83 PCB. The value sent is the 6 lowest bits of the DIP-switch setting, incremented by 30H to get an ASCII value (30H to 6FH)
- The LRC is calculated by XOR-ing all the bytes in STRING and OR-ing the result with 20H (initial value is 00H). This will ensure that the LRC is not misinterpreted as an ASCII control character (they are all less than 20H).

4.1.2 OFF- line mode

If the controller doesn't get an ACK after transmitting a record in ON-LINE mode, it will go into the OFF-LINE state in which it will store the clocking records in its buffer. Approximately 60 seconds after no clockings on either of the TK0 or TK1 channels, it will try to transmit a block of records by sending a so called 'service request' string (<SOH> <'V'>). If the PC answers, the controller will transmit a block of data. After a while it will send another 'service request ' string, transmit a block and so on, until all records in the buffer are transmitted. At that point it will return into the ON-LINE mode and transmit recordss as they happen.

SL-83⇨PC

<SOH> <'V'>

PC⇨SL-83

<ACK> if the PC is ready to receive the data

If it isn't ready, nothing is sent, or if a different character is received (not ACK) the controller stops its transmission request. It will try again after a fixed time.

If an <ACK> is received the transmission continues.

SL-83⇨PC

<STX>

(STRING) <CR>

(STRING) <CR>

.....

(STRING) <CR>

<ETX>

<LRC>

If the PC receives all of the data correctly, an <ACK> is sent to the controller.
(The timeout for this ACK is greater than in the other cases because the PC may need more time to process the block and write the data to disk.)

PC⇨SL-83

<ACK>

Notes:

- The LRC is calculated by XOR-ing the bytes in the STRING and OR-ing the result with 20H.
- The number of records [(STRING)<CR>] in a block can be between 1 and 32. It depends on whether the previous block was successfully received and ACK-ed. A bad reception, indicated by a NAK, will result in the length of the block being decreased, while a good reception will increase it. Thus, the block length is adapted to the quality of the communication line.

4.2. Terminal commands

S	Status	read the status of the controller
T	Send	tells the controller to start sending data
D	SetTime	copy current PC time to the controller's clock
P	PackData	Pack clocking data
G	GlobalMessage	Send a string of 24 characters to the controller. Displayed on lower row of LCD
M	ClearMessage	delete string sent by GlobalMessage

The protocol of these commands is as follows:

PC⇨SL-83

<SOH>

SL-83⇨PC

<ACK> (if the controller is not busy)
(if it is busy, it doesn't respond at all, so the PC has to repeat the command later)

The later course of communication depends on the type of command. The simplest are the ones without any parameters.

4.2.1 Terminal commands without parameters

These commands consist of one ASCII character, which is sent twice to minimise errors.

PC⇨SL-83

<CHAR> <CHAR>

If the controller recognises the received character as a valid command (from the set of command characters), and if both received characters are the same, it replies with an <ACK> to the PC and executes the command. Otherwise it returns a <NAK>.

The commands can be one of the following:

‘S’ Status

Makes the controller send a status string. This string contains the following:

- current time and date according to the controller’s clock
- number of clocking records in the terminal’s buffer
- firmware (EPROM) version
- DIP switch setting
- Reset and Fatal Reset counters

‘T’ Transfer (start sending records if there are any in the buffer)

Initiates a transmit sequence straight away. This sequence is usually initiated by the terminal itself a certain period after no clockings have occurred. (This scheme is used to minimise delays for people using the controller.) If the operator of the master PC has a specific reason not to wait for that event, he can initiate transmission immediately with this command.

This can be useful, for example, if the PC was off or the acquisition program inactive and the user wishes to collect the accumulated data as quickly as possible and then generate a report or switch the PC off again.

‘M’ Erase message

This will erase the message displayed on the lower row of the LCD, which was previously put there by the GlobalMessage command.

4.2.2 Terminal commands with parameters

‘D’ Set Date and Time

After receiving the <ACK> from the controller, the PC sends a string of 15 characters to the controller
:
<D10><D1><S10><M10><M1><S1><Y10><Y1><DW><H10><H1><:><m10><m1><LRC>

Explanation:

<D10>	:	10s digit of the day field (ASCII character: 30H, 31H, 32H or 33H)
<D1>	:	1s digit of the day field
< S10 >	:	10s digit of the seconds field
<M10>	:	10s digit of the month field
<M1>	:	1s digit of the month field
<S1>	:	1s digit of the seconds field
<Y10>	:	10s digit of the year field
<Y1>	:	1s digit of the year field
<DW>	:	day of week (30H – Sunday, 31H – Monday, …, 36H – Saturday)
<H10>	:	10s digit of hours field
<H1>	:	1s digit of hours field
< : >	:	3AH
<M10>	:	10s digit of minutes field
<M1>	:	1s digit of minutes field
<LRC>	:	calculated by XOR-ing all transmitted characters and OR-ing the result with 20H

If the controller receives the string without errors it sends an <ACK> and sets its clock according to the data.

Otherwise it sends a <NAK>.

Note: in the current FW version (8303x), seconds are not used for clockings. The point of sending the second data is better synchronisation of multiple controllers in a single system.

‘P’ PackData

This command has the effect of checking the whole of the circular buffer (irrespective of the RD and WR pointers) for valid records (with good checksums) . It then stacks all the good records it finds from the start of the buffer and increments the new WR pointer. This has the effect of reviving all records in the buffer, even those that have already been downloaded to the PC.
The command should be used when the terminal hangs or is reset in such a way that the pointers are corrupted (due to spikes, lightning etc.) and a FATAL reset occurs. It is quite possible that the data itself is still in order (at least some of it) and only the pointers have been lost.
The packing operation can last a few minutes and the controller does not respond to commands or events in that time.

4.2.3 Access table related commands

These commands are used to edit the access tables, i.e. the tables that define the users: ID table, DayTypes and WeekTypes.

e	KillIDElement	Delete a code from the ID table
i	AddIDElement	Add a new code to the ID table
j	IDT_Num	Sends current number of codes in ID table
k	IDT_Clear	Clears whole ID table
r	Reset IDT_CurrTx	Resets table pointer
s	TxCurrITD	Sends current element of ID table
t	RxDayType	Receives new DayType
u	RxWeekType	Receives new WeekType
v	TxDayType	Sends current DayType
w	TxWeekType	Sends current WeekType

General notation conventions:

‘CH’ - ASCII code is transmitted e.g. ‘0’ = 30H
<CH> - denotes that a single ASCII character is transmitted e.g. <ACK> = 06H
(CH) - the byte is transmitted as two ASCII characters i.e.:
[16] = [10H]= 31H 30H

‘e’ KillIDElement

Delete specified element from the ID table.

PC⇨SL-83

(FC)(B5)(B4)(B3)(B2)(B1)(B0)(AB) <ETX> <LRC>

SL-83⇨PC

reception OK:

<ACK> ‘0’ element deleted from table
<ACK> ‘1’ element not deleted (not found)

reception error:

<NAK>

Notes:

(FC) Family Code byte of the iButton (44H in case of RF tags)
(B5)...(B0) ID bytes of the badge
(AB) ActionByte (user defined)
<LRC> calculated by XOR-ing all bytes in the STRING and OR-ing the result with 20H

‘i’ AddIDElement

PC⇨SL-83

(FC) (B5)(B4)(B3)(B2)(B1)(B0)(AB) <ETX> <LRC>

SL-83⇨PC

Reception OK:

<ACK> '0'	new element added to the table
<ACK> '1'	new element not added (table full)

reception error:

<NAK>

Notes:

(FC)	Family Code of the iButton i.e. 01H for DS1990A.
(B5)...(B0)	ID code bytes of the iButton
(AB)	ActionByte (user define – determines access rights)
<LRC>	control character calculated by XOR-ing the bytes of the string and OR-ing this with 20H

'k' IDT_Clear

Clears the whole table of ID codes. (Internally only the ActionByte of each table element is cleared).

'r' Reset IDT_CurrTx

Resets the IDT_CurrTx variable, which serves as a pointer to the access tables and determines which entry will be sent with the next command for transmitting table entries. Each command for sending entries increments this pointer by one. It should be noted that since the same pointer is used for all of the three tables, it is not possible to transmit elements of different tables in an interleaved fashion. A whole table should be sent, then the pointer reset and only then should one start with another table.

's' TxCurrIDT

Sends the currently indexed (by the IDT_CurrTx pointer) element of the ID table. After receiving an <ACK> from the PC the IDT_CurrTx pointer is incremented.
If the IDT_CurrTx > 4095 only an <EOT> is sent.

SL-83⇨PC

n..n '/' (FC)(B5)(B4)(B3)(B2)(B1)(B0)(AB)<ETX> <LRC>

Notes:

n..n	IDT_CurrTx in decimal notation (ASCII characters, variable number of digits),
'/'	separator = ASCII 2FH,
<LRC>	this byte is calculated by XOR-ing all the bytes of the sent string (excluding ETX) and OR-ing the result with 20H.

't' RxDaytype

Upload a new element of the DayType table:

PC⇨SL-83

<n> <FH1> <FM1> <TH1> <TM1> <FH2> <FM2> <TH2> <TM2> <ETX> <LRC>

SL-83⇨PC

Transmission OK:

<ACK> '0'	element added to table,
<ACK> '1'	<n> out of range (0 .. 31).

Transmission error:

<NAK>

Notes:

- <n> - position of element in table (0 to 31)
- the LRC is calculated by XOR-ing all of the received characters from <n> to <ETX> (including them). The result is OR-ed with 20H to ensure an ASCII character greater or equal than 20H).
- the timeout period for the ACK from the PC is around 500 ms. If it doesn't arrive in that time the transmission is considered erroneous and IDT_CurrTx is not incremented, which means that the next 'w' command will cause the same table element to be sent.

'u' RxWeekType

Uploads a new element of the WeekType table.

PC⇨SL-83

<n><DT0><DT1><DT2><DT3><DT4><DT6><NZJ><ETX><LRC>

SL-83⇨PC

transmission OK:

<ACK>'0'	element added to table,
<ACK>'1'	<n> out of range (0 to 31),

transmission error:
<NAK>

Notes:

- <n> position of element in table (0 to 31)
- <NU> not used

- the LRC is calculated the usual way
- the timeout period for the ACK from the PC is about 500 ms. If it doesn't arrive in this time span the transmission is erroneous and the IDT_CurrTx is not incremented, which means that the next 'u' command will cause the same WeekType table element to be sent.

'v' TxDayType

Downloads the currently selected (by the pointer IDT_CurrTx) element of the DayType table, after receiving an ACK from the PC the IDT_CurrTx is incremented.

If the IDT_CurrTx is greater than 31 only an <EOT> is sent.

For IDT_CurrTx < 32:

SL-83⇨PC

<n><FH1><FM1><TH1><TM1><FH2><FM2><TH2><TM2><ETX><LRC>

PC⇨SL-83

Transmission OK:
<ACK>
(IDT_CurrTx ▢ IDT_CurrTx + 1)

transmission error:
<NAK>

For IDT_CurrTx > 31:

SL-83⇨PC

<EOT>

Notes:

- the LRC is calculated in the usual way

- If the PC doesn't send an ACK during the timeout period of 500 ms the transmission is erroneous and the IDT_CurrTx pointer is not incremented. The next 'v' command will cause the transmission of the same element of the WeekType table.
- During transmission the following appears on the lower row of the LCD:
  TxDyt: XX
  where XX represents the number of the currently transmitted element (IDT_CurrTx).

'w' TxWeekType

Downloads the currently selected (by the pointer IDT_CurrTx) element of the WeekType table, after receiving an ACK from the PC, increments the IDT_CurrTx pointer.

If IDT_CurrTx > 31 only an <EOT> is sent.

For IDT_CurrTx < 32:

SL-83⇨PC

<n><DT0><DT1><DT2><DT3><DT4><DT5><DT6><NU><ETX><LRC>

PC⇨SL-83

Reception OK:
<ACK>
(IDT_CurrTx = IDT_CurrTx + 1)

Reception error:
<NAK>

For IDT_CurrTx ≥ 32:
SL-83⇨PC

<EOT>

Notes:
- LRC is calculated by XOR-ing all the sent bytes from <n> to <NU> (not including ETX) . This value is OR-ed with 20H to ensure that the obtained byte will always be an ASCII character (>= 20H).
- The timeout for the PC's ACK is about 500 ms. If it doesn't arrive in that time, the transmission is considered erroneous and the IDT_CurrTx is not incremented. This means that the next 'w' command will cause the same element of the WeekType table to be sent.
- During transmission the lower line of the LCD shows the following:
  TxWkt: XX
  where XX is the number of the currently transmitted (IDT_CurrTx) element in hexadecimal.

4.2.4 Commands related to the configuration parameters

The configuration parameters are stored in the serial EEPROM. These commands are really only simple EEPROM related commands, with which one can read and write any word (16-bit) of the serial EEPROM.
The size of the currently used EEPROM is 256x16 bits. But this may change in the future.

'a' ReadEEPR

Downloads the word (16-bit) at the specified address of the serial EEPROM.
After receiving an ACK from the terminal, the following is sent:

PC⇨SL-83

(Adr)<LRC>

Reception OK:

SL-83⇨PC

<ACK>
(WordH)(WordL)<LRC>

Reception error:

SL-83⇨PC

<NAK>

Notes:

- (Adr) is the address of the EEPROM word to be read. It is in hexadecimal notation for 256x16 EEPROMS it can have a value between ‘00’ and ‘FF’
- (WordH) and (WordL) represent the high and the low byte of the returned 16-bit word
- The <LRC> characters are calculated in the usual way

‘b’ WriteEEPR

Uploads (writes) the specified 16-bit word to the specified address of the EEPROM.
After receiving an ACK from the terminal, the following is sent:

PC⇨SL-83

(Adr)(WordH)(WordL)<LRC>

Reception OK:

SL-83⇨PC

<ACK>

<ACK>

Reception error:

SL-83⇨PC

<NAK>

Notes:
The second <ACK> is sent by the controller to indicate when the write operation has been completed. This is important in multiple write operations because an EEPROM write operation can take up to 10ms.

5. Configuration

The SL-83 controller can be configured by the user, this means that the administrator can change the way the controller behaves in certain situations. This programmability is necessary to adapt the device to various applications.
Programmability of the SL-83 is achieved by allotting a certain number of bits in the EEPROM to each possible event. This, so called ActionWord (not to be mistaken with the ActionByte of the ID table) contains coded information that tells the controller what actions to take in which situations. These events are as follows:

Event ID	Event code	Event source	Description	Clocking data
T1	0	ID contact1	iButton touched and successfully read	ID code
TA1	0	ID contact1	iButton code in ID table, AA bit set	ID code
TB1	0	ID contact1	iButton code in ID table, AB bit set	ID code
TC1	0	ID contact1	iButton code in ID table, AC bit set	ID code
T2	1	ID contact2	iButton touched and successfully read	ID code
TA2	1	ID contact2	iButton code in ID table, AA bit set	ID code
TB2	1	ID contact2	iButton code in ID table, AB bit set	ID code
TC2	1	ID contact2	iButton code in ID table, AC bit set	ID code
EC1	4	AUX unit	Extended ID code #1	ID code
EC2	5	AUX unit	Extended ID code #2	ID code
EC3	6	AUX unit	Extended ID code #3	ID code
EC4	7	AUX unit	Extended ID code #4	ID code
EC5	8	AUX unit	Extended ID code #5	ID code
EC6	9	AUX unit	Extended ID code #6	ID code
EC7	A	AUX unit	Extended ID code #7	ID code
EC8	B	AUX unit	Extended ID code #8	ID code
EC9	C	AUX unit	Extended ID code #9	ID code
EC10	D	AUX unit	Extended ID code #10	ID code
EC11	E	AUX unit	Extended ID code #11	ID code
S1O	F	Sensor input1	Door opened	00000080
S1C	F	Sensor input1	Door closed	00000081
S2O	F	Sensor input2	Door opened	00000082
S2C	F	Sensor input2	Door closed	00000083
E1O	F	Egress input1	Egress contact open	00000088
E1C	F	Egress input1	Egress contact closed	00000089
E2O	F	Egress input2	Egress contact open	0000008A
E2C	F	Egress input2	Egress contact closed	0000008B
AC1F	F	AC voltage1	AC voltage failed	00000090
AC1O	F	AC voltage1	AC voltage OK	00000091
AC2F	F	AC voltage2	AC voltage failed	00000092
AC2O	F	AC voltage2	AC voltage OK	00000093
JUMPO	F	Jumper	Jumper open	00000094
JUMPC	F	Jumper	Jumper closed	00000095
AFE1	F	Sensor input1	Forced entry: door1 opened/ relay1 off	00000096
AFE2	F	Sensor input2	Forced entry: door2 opened/ relay2 off	00000097
ADOTL1	F	Sensor input1	Door1 open too long	0000009A
ADOTL2	F	Sensor input2	Door2 open too long	0000009B
TIMEV1	F		Time event #1 occured	0000009E
TIMEV2	F		Time event #2 occured	0000009F
TIMEV3	F		Time event #3 occured	000000A0
TIMEV4	F		Time event #4 occured	000000A1
TIMEV5	F		Time event #5 occured	000000A2
TIMEV6	F		Time event #6 occured	000000A3
TIMEV7	F		Time event #7 occured	000000A4
TIMEV8	F		Time event #8 occured	000000A5

RNDM1	F		Random event #1 occured	000000A6
RNDM2	F		Random event #2 occured	000000A7
DIRCM1	F		Direct command #1 received	000000AA
DIRCM2	F		Direct command #2 received	000000AB
DIRCM3	F		Direct command #3 received	000000AC
DIRCM4	F		Direct command #4 received	000000AD
DIRCM5	F		Direct command #5 received	000000AE
DIRCM6	F		Direct command #6 received	000000AF
DIRCM7	F		Direct command #7 received	000000B0
DIRCM8	F		Direct command #8 received	000000B1
	F			

Notes

- The ID of the events are just a short way of notation (i.e. T1, TB2, S2C) while the event code is what gets stored in a record's activity field, if clocking is enabled.
- Some events like T1 and TC1 have the same code. This can lead to ambiguities if clocking is allowed for both events T1 and TC1. The reason for wanting both events to generate clockings can be that we want to have a record of all the badges that tried to gain access but were not granted it. If a badge was used to gain access (and was active at the time) we will have two consecutive clockings in the ASCII file.

5.1. Description of the ActionWord

The following actions are defined for each event:

- How are the relay outputs affected? (RL1 and RL2)

It is possible to define one of 4 options for each relay:

1. No effect on relay (---)
2. Relay turned off (OFF)
3. Relay turned on for preset time (RLT) (see RELAYTIME configuration parameter)
4. Relay turned on indefinitely (ON)

This is the so-called 'relay block' of bits in the ActionWord. There are four such relay blocks in each ActionWord, one for each of the relays.

- Conditional block

The preceding four possible actions to be taken with the relay is further made dependent on certain conditions

1. No condition (.i.e. unconditionally execute the corresponding 'relay block')
2. Do 'relay block' if the sensor input is open (Sn=0) (n=1,2)
3. Do 'relay block' if the sensor input is shorted (Sn=1) (n=1,2)
4. Do 'relay block' if the sensor input is in the same state as the egress input (Sn=En) (n=1,2)

- Is a record made? (sent immediately to the PC or written to the buffer)

1. record made (REG)
2. record not made (NRG)

Bits of the ActionWord

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

RL2 block

RL1 block

RL2 condition

RL1 condition

Record

00	Always
01	S2=0
10	S2=1
11	S2=E2

00	Always
01	S1=0
10	S1=1
11	S1=E1

0	NRG
1	REG

00	No Effect
01	Off
10	On RLT
11	On

Bits B15, B14 and B11 are currently not used and are set to B15='1', B14='0' and B11='0'

Bits B7,B6,B5 and B4 are also not used and their value is unimportant.

Example:

An event occurs and the corresponding ActionWord is read out from the EEPROM. It has the following value:

10010111 00011011b. This will have the following effects:

- RL1 will be activated IF S1=E1 (sensor1 input = egress1 input)
- RL2 will be activated for the configured RLT (relaytime) IF S2=1 (sensor2 input is shorted)
- A record will be made

5.2. Other configuration parameters

Apart from ActionWords for each of the possible events, there are other types of parameters stored in the configuration EEPROM.

5.2.1. LED polarity

The LED indicators on the TK0, TK1 iButton readers have the following mode of operation: by default, the LEDs are off and are turned on for a few seconds when the iButton is read (green) and when the corresponding relay is activated (red or yellow).
Because of the need to use reader modules with onboard electronics (iButton reader module with PIN keyboard, magcard reader module, RF reader module etc) it is necessary to be able to change the polarity of the LED driver outputs. By doing this, and keeping the outputs high by default we can use these outputs as "phantom" power supply lines for the electronics.

The LEDCNF location in the serial EEPROM at address 38 (26H) is defined like this:

G1 (Green) defines the behaviour of the green LED of TK0 #1 and comprises two bits (b15 & b14) of the 16-bit word.

00 default state: voltage on, 2 sec. off in case of activation
01 default state: voltage on, 1 sec. off in case of activation
10 default state: voltage off, 2 sec. on in case of activation
11 default state: voltage off, 1 sec. on in case of activation

The rest of the LEDs are defined in the same way, only the position of the two bits in LEDCNF is different.

If ordinary iButton readers are used, either state 10 or 11 can be selected, but for readers with internal electronics state 01 is best because upon activation of the LEDs, the module is left without a supply voltage for a shorter time than with state 00. The bridging capacitors can be smaller as a consequence.

The PC program for configuration (Cnf8303.exe) has a tab for setting the LED polarities.
States 00 to 11 are displayed graphically, as voltage levels, so selection is much more straightforward.

LEDCNF 74 (decimal)

15	14	13	12	11	10	9	8	7	6	5	b	3	2	1	0
G1	G1	R1	R1	G2	G2	R2	R2	G3	G3	R3	R3	G4	G4	R4	R4
H	L	H	L	H	L	H	L	-	-	-	-	-	-	-	-

5.2.2. Random events

Some companies require the generation of random events as a result of in/out clocking of employees. This signal is used to generate a visual or audio signal for searching the employee or doing an alcohol breath-test etc.

Random events are triggered by the reading a badge on either of the 2 ID modules and each of the ID channels (1,2) generates a different random event configured in its separate ActionWord.

The EEPROM addresses of the random event ActionWords are:

ActionWord	EEPROM addr.
RANDEV1	170 (decimal)
RANDEV2	172 (decimal)

The probability i.e. frequency of the generation of these events can be set between 0 (never) and 255 (always).

The random event generator uses a fast revolving 8-bit counter, which is read out at the moment a badge is read. If the counter state is less than the preset frequency value, the corresponding ActionWord is executed.

Each one of the two ID modules can be active for random event generation. Sometimes it is only necessary, for instance, to generate random events for the outgoing employees (spot searches) and sometimes only for the incoming ones (breath tests).

The high byte is used for the 8-bit random event frequency, the lower four bits of the lower byte hold the 'active'/'not active' bits of the four contact plaques.
A 1 denotes active, a 0 not active.

RANDOM (76 decimal)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F7	F6	F5	F4	F3	F2	F1	F0							C2	C1

5.2.3. Relay On Time (RELAYTIME)

This parameter sets the relay on time for each of the 2 relay outputs.

RELTIM (99 decimal)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R43	R42	R41	R40	R33	R32	R31	R30	R23	R22	R21	R20	R13	R12	R11	R10

The values of the 4-bit nibbles are interpreted into relay times according to the following table:

Value (bin)	Time
X000	1s
X001	2s
X010	4s
X011	8s
X100	16s
X101	32s
X110	1 min.
X111	2 min.

These times will apply for all ActionWords that specify a relay being active for RLT (relaytime) period of time.

5.2.4. Environment Events

Environment events enable the user to log and perform certain actions if the environment of the terminal changes, e.g. the case door is opened or the AC supply voltage disappears or re-appears (the terminal can function with its batteries for a certain period). Each environment event has its own ActionWord in the EEPROM.

Event	Description	Registration data	EEPROM address
AC1F	AC1 voltage failed	00000090	90 (decimal)
AC1O	AC1 voltage OK	00000091	92 (decimal)
AC2F	AC2 voltage failed	00000092	94 (decimal)
AC2O	AC2 voltage OK	00000093	96 (decimal)
JUMPO	Jumper open (cover)	00000094	86 (decimal)
JUMPC	Jumper closed (cover)	00000095	88 (decimal)

- AC1 and AC2 are now the same event. The two different events were necessary because previous versions of the SL-83 used a transformer with two secondary windings. The SL-83R7 uses a single-output switching regulator and an onboard DC/DC converter, so there is only one input for checking external input voltage and this is connected to both AC1 and AC2 inputs.

- Jumper JP2 (JUMPO, JUMPC) turns the LED backlight of the LCD on/off automatically and can be used with a microswitch to monitor the state of the cover (lid) of the SL-83 metal case.

5.2.5. Time Events

Time events can be set to occur only once or at regular intervals (every year, month, date, day of week, hour, minute). Each of the 8 time events has its own ActionWord in the EEPROM.

Event	EEPROM address
TIMEEVENT1	110 (decimal)
TIMEEVENT2	112 (decimal)
TIMEEVENT3	114 (decimal)
TIMEEVENT4	116 (decimal)
TIMEEVENT5	118 (decimal)
TIMEEVENT6	120 (decimal)
TIMEEVENT7	122 (decimal)
TIMEEVENT8	124 (decimal)

Time event
ActionWord addresses

Apart from the ActionWord addresses given in the previous table, each time event has a setup block address in the EEPROM, which defines when the event will occur.

Setup block	EEPROM address
TIMESETUP1	126 (decimal)
TIMESETUP2	130 (decimal)
TIMESETUP3	134 (decimal)
TIMESETUP4	138 (decimal)
TIMESETUP5	142 (decimal)
TIMESETUP6	146 (decimal)
TIMESETUP7	150 (decimal)
TIMESETUP8	154 (decimal)

Time event
Setup block addresses
addresses

Each setup block consists of four 16-bit words. The contents are shown in the following table:

Addr. Offset	High byte	Low byte
0	year	month
1	day	hour
2	minute	day of week
3	csum	ncsum

Notes:

- Day of week values are between 0 (Sunday) and 6 (Saturday)
- Year values are between 00 and 99

- Month values are between 01 and 12
- Day values are between 01 and 31
- Hour values are between 00 and 23
- Minute values are between 00 and 59
- Any one of the previous fields can be set to A5H, which means that it's a ***don't care*** value.
  E.g. if all the time fields were set to A5h, except the minutes field, which was set to 00, the time event would occur at every full hour.
- Csum is a checksum field, calculated by 8-bit addition of the time fields (high byte of offset 0 + low byte of offset 0 + high byte of offset 1 + … + low byte of offset 2)
- Ncsum is the 1's complement of csum

5.2.6. Forced Entry Alarm

An event can be generated if one of the (door) sensor inputs is opened without the corresponding relay output being active. This implies that the door was opened with a key or physical force, circumventing the SL-83.

"Corresponding" relay output means the same channel number, i.e. sensor #1 must be used with relay #1 and so on.

Each one of the 2 available sensor inputs can generate its own Forced Entry Alarm, with its own ActionWord in the EEPROM.

Event	**EEPROM address**
AFE1	78 (decimal)
AFE2	80 (decimal)

5.2.7. Door Open Too Long Alarm

An event can be generated if one of the (door) sensor inputs is held open for too long. Each one of the 2 door sensors can be set to a different time limit for the event to occur.

Event	**EEPROM address**
ADOTL1	100 (decimal)
ADOTL2	102 (decimal)

The time limits are stored in the following location:

DOOROPN (98 decimal)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
-	-	-	-	-	-	-	-	D23	D22	D21	D20	D13	D12	D11	D10

D13, D12, D11, D10 are bits of the 4-bit value for sensor#1 time limit
D23, D22, D21, D30 are bits of the 4-bit value for sensor#2 time limit

The correspondence between the values of the time limit nibbles and the actual time in seconds is approximately such, that a value of 01 gives a 1s time, 02 a 2s time etc. up to 0F (15 sec.).

5.2.8. AUX cyclic functions

Each of the 8 available AUX cyclic functions has a one-word (16-bit) definition in the EEPROM, which looks like this:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	F2	F1	F0	n.u.	n.u.	MC2	MC1	MC0

- A7..A0 is the AUX address of the unit performing the cyclic function
- F2..F0 is the AUX function number (0..7)
- MC2..MC0 is the MAXCOUNT value, i.e. this defines the frequency with which the cyclic function will be called by the SL-83 terminal. A translation table is used to convert this 3-bit value into a counter for the for main loop cycles of the SL-83 terminal, which has a unit value between 8 and 10ms.

AUX function	EEPROM address
AUX1	162 (decimal)
AUX2	163 (decimal)
AUX3	164 (decimal)
AUX4	165 (decimal)
AUX5	166 (decimal)
AUX6	167 (decimal)
AUX7	168 (decimal)
AUX8	169 (decimal)

AUX cyclic functions
Setup addresses

5.2.9. AUX ID Event table

This table lets the user assign ActionWord event blocks to certain clockings received from AUX units. The clockings are defined by AUX address/ Event Code combinations.

Note: the term 'event block' denotes the execution of one or more of four ActionWords, depending on whether the ID code is in the ID table and which of the three ActionBits are set:
i

- For all clockings
- For clockings whose ID field is found in the ID table and has an A-bit set
- For clockings whose ID field is found in the ID table and has a B-bit set
- For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	EC3	EC2	EC1	EC0	n.u.	n.u.	IE1	IE0

- A7..A0 is the AUX address of the unit which sent the clocking
- EC3..EC0 is the Event Code sent in the clocking
- IE1..IE0 is the AUX ID event block (0,1,2,3) ssociated to this Address/Event Code combination

AUX IDE table	EEPROM address
AITT1	234 (decimal)
AITT2	235 (decimal)
AITT3	236 (decimal)
AITT4	237 (decimal)
AITT5	238 (decimal)
AITT6	239 (decimal)
AITT7	240 (decimal)
AITT8	241 (decimal)

AUX ID Event Translation
Table addresses

5.2.10. AUX ID Default Event block

This is an event block of four ActionWords that are executed (depending on whether the ID code is in the ID table and the setting of the Action Bits) if the Address/Event Code combination is not in the AUX ID Event table. In other words, this event block 'catches' all the AUX ID clockings not specifically entered into the AUX ID Event table.

ActionWord	EEPROM address
AIED	194 (decimal)
AIEDA	196 (decimal)
AIEDB	198 (decimal)
AIEDC	200 (decimal)

This event block has four standard ActionWord definitions which are executed:

- **AIED** For all clockings
- **AIEDA** For clockings whose ID field is found in the ID table and has an A-bit set
- **AIEDB** For clockings whose ID field is found in the ID table and has a B-bit set
- **AIEDC** For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

5.2.11. AUX ID Event 1

This is an event block of four ActionWords that are executed (depending on whether the ID code is in the ID table and the setting of the Action Bits) if the received Address/Event Code combination is associated to AUX ID Event 1.

ActionWord	EEPROM address
AIE1	202 (decimal)
AIE1A	204 (decimal)
AIE1B	206 (decimal)
AIE1C	208 (decimal)

This event block has four standard ActionWord definitions which are executed:

- **AIE1** For all clockings
- **AIE1A** For clockings whose ID field is found in the ID table and has an A-bit set
- **AIE1B** For clockings whose ID field is found in the ID table and has a B-bit set
- **AIE1C** For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

5.2.12. AUX ID Event 2

This is an event block of four ActionWords that are executed (depending on whether the ID code is in the ID table and the setting of the Action Bits) if the received Address/Event Code combination is associated to AUX ID Event 2.

ActionWord	EEPROM address
AIE2	210 (decimal)
AI2A	212 (decimal)
AIE2B	214 (decimal)
AIE2C	216 (decimal)

This event block has four standard ActionWord definitions which are executed:

- **AIE2** For all clockings
- **AIE2A** For clockings whose ID field is found in the ID table and has an A-bit set
- **AIE2B** For clockings whose ID field is found in the ID table and has a B-bit set
- **AIE2C** For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

5.2.13. AUX ID Event 3

This is an event block of four ActionWords that are executed (depending on whether the ID code is in the ID table and the setting of the Action Bits) if the received Address/Event Code combination is associated to AUX ID Event 3.

ActionWord	EEPROM address
AIE3	218 (decimal)
AIE3A	220 (decimal)
AIE3B	222 (decimal)
AIE3C	224 (decimal)

This event block has four standard ActionWord definitions which are executed:

- **AIE3** For all clockings
- **AIE3A** For clockings whose ID field is found in the ID table and has an A-bit set
- **AIE3B** For clockings whose ID field is found in the ID table and has a B-bit set
- **AIE3C** For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

5.2.14. AUX ID Event 4

This is an event block of four ActionWords that are executed (depending on whether the ID code is in the ID table and the setting of the Action Bits) if the received Address/Event Code combination is associated to AUX ID Event 4.

ActionWord	EEPROM address
AIE4	226 (decimal)
AIE4A	228 (decimal)
AIE4B	230 (decimal)
AIE4C	232 (decimal)

This event block has four standard ActionWord definitions which are executed:

- **AIE4** For all clockings
- **AIE4A** For clockings whose ID field is found in the ID table and has an A-bit set
- **AIE4B** For clockings whose ID field is found in the ID table and has a B-bit set
- **AIE4C** For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

6. AUX Interface

The AUX interface is a 4-wire bidirectional interface for connecting peripheral units to the SL-83 controller. The protocol is very similar to the I²C bus, with the two other wires being GND and V+.

Vertical RJ-11 connector pinout:

1	V+	12V-13,8V, 0,9A max
2	CLK	SCL equivalent
3	DATA	SDA equivalent
4	GND	

The AUX modules can be daisy-chained on the AUX interface, each having an address which is defined by a pre-set base address (for that type of device) and one or more jumpers or DIP switches for setting the sub-addresses.

Each AUX unit has two 4-pin RJ-11 connectors, one for the incoming cable (from the master i.e. terminal), the other towards the following unit in the daisy-chain.
The last (and only the last) unit in the chain must have the two jumpers for pull-up resistors set.

Current firmware on the SL-83 (8303) supports up to 8 different AUX units connected to a master.

6.1. Support for AUX units in the SL-83R7 firmware

There are two modes of firmware support for AUX devices:

- Cyclic functions
- Commands

Cyclic functions are are carried out regularly (every certain number of cycles of the main SL-83 firmware loop) and are defined as follows:

The **address** of the unit defines the type of AUX unit (due to the range of addresses allotted to specific types of AUX units). The firmware of the SL-83 controller 'knows' how to handle AUX units based on their addresses. As new AUX units are made available, the firmware will be updated to handle the new devices.

Certain AUX units have more than one command that can be carried out cyclically. In this case, these commands will be given **function** numbers. Thus, the user can define which commands of the AUX unit in question will be called and with which frequency.

The correspondence between (address,function no.) pairs and what this means in terms of commands to the AUX unit is fixed in the SL-83 firmware.

Example:

The SL-870 is an auxiliary ID terminal, that is attached to the AUX chain. The address range is 30H to 33H. Two cyclic operations can be identified and allotted numbers:

- Function 00: the SL-83 sends the current time & date to the SL-870 for it to display accurate time & date data on its display. (The SL-870 hasn't got its own clock chip.)
- Function 01: the SL-83 requests clocking data (ID code and activity) to be sent.

Function 00 can be carried out once every 10 seconds, since the SL-870 can update the clock data by itself for short periods, whereas function 01 has to be done at much smaller intervals (under 1s) so the

person who's just made a clocking sees immediate confirmation from the SL-870 and the next person can make a clocking.

Commands are carried out as a result of commands received by the SL-83 from a master PC. They are usually done as a result of human intervention: status read or configuration of an AUX unit.
Commands are received by the SL-83 and put into the AUX queue ahead of cyclic functions. There is no queue for multiple AUX commands, so the previous command has to be finished before a new one can be received.

6.1.1. AUX tree of the CNF8303.EXE setup program

There are four types of tables in the AUX section of the configuration program:

- Aux functions -defines up to 8 AUX cyclic functions
- Aux ID Event table -defines which AUX address/action will cause which ID event block
- Aux ID Event default -defines the default AUX ID event block
- Aux ID Event #1 to #4 -defines the 4 possible ID event blocks

6.1.2. Aux functions

This node defines the cyclic AUX functions There can be a maximum of 8 cyclic functions and, depending on the type uf AUX units used, a maximum of 8 AUX devices i.e. addresses.

Each row has the following options:

Active
Checkbox. Enables you to quickly enable or disable an AUX unit or a function of a multi-function unit.

Address
This is the unique address of the AUX unit which performs the desired function.

Function
The function being performed. In single-function units, this field is not relevant. Multi-function units must have each used function defined in a separate row of this table.

Frequency
A number between 0 and 7 indicating the cycle rate of the function. The frequency/cycles relationship isn't linear but defined according to the following table:

Frequency	0	1	2	3	4	5	6	7
Cycles	1	2	3	4	10	20	100	255
Time	8ms	16ms	25ms	30ms	90ms	170ms	800ms	2s

6.1.3. AUX ID Event Table

This table lets the user assign standard ActionWord events to certain clockings received from AUX units. The clockings are defined by AUX address/ Event Code combinations.

Note: you can't have more than one ID Event assigned to a single Address/Event Code combination. In other words, If there are two rows in the AUX ID Event table with the same Address/Event code combination, only the first one will be carried out.

Active
Checkbox. Enables you to quickly enable or disable an AUX ID Event Table entry

Address
This is the address part of the Address/Event code pair

Event Code
This is the Event Code part of the Address/Event code

ID Event
This is one of four AUX ID events which is assigned to this Address/Event Code combination

6.1.4. AUX ID Event Default

This is the default event that is carried out if the Address/Event Code combination is not found in the AUX ID Event table.

The tab has four standard ActionWord definitions which are executed:

- For all clockings
- For clockings whose ID field is found in the ID table and has an A-bit set
- For clockings whose ID field is found in the ID table and has a B-bit set
- For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

6.1.5. AUX ID Event 1 ... AUX ID Event 4

These tables define the ActionWords (events) that should be executed when an AUX ID clocking arrives which is assigned to one of these events via the AUX ID Event table.

The tabs have four standard ActionWord definitions which are executed:

- For all clockings
- For clockings whose ID field is found in the ID table and has an A-bit set
- For clockings whose ID field is found in the ID table and has a B-bit set
- For clockings whose ID field is found in the ID table and has a C-bit set and the time/date condition is met

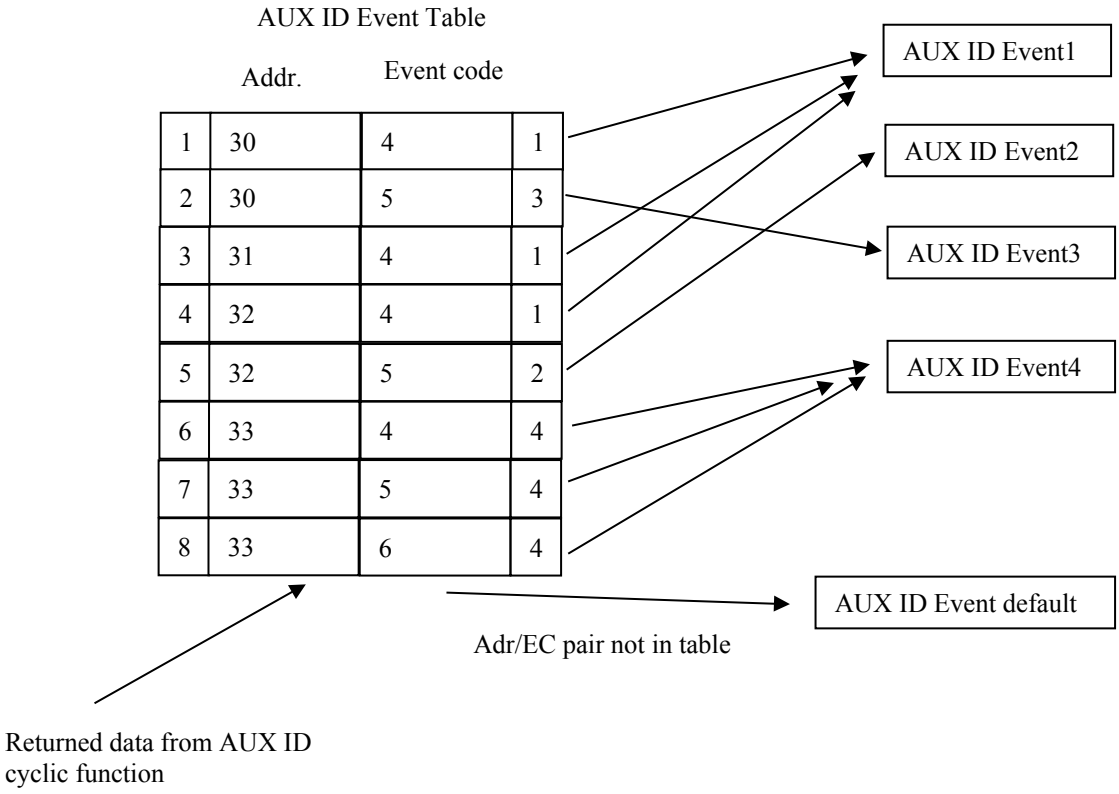
Example:

In the example below, all four of the available AUX ID events are used and the ID events are expected from SL-870 ID terminals. These terminals are primarily meant for time&attendance recording and can't generate an event for access control (e.g. activating a relay) directly. With these AUX ID events, however, we can associate regular ActionWords to the AUX ID clockings arriving from the 4 SL-870 units.

The first row in the AUX ID Event table below indicates that the event code 04, in a clocking coming from the SL-870 unit with the address 30h is associated with AUX ID Event #1. If Event #1 is configured to activate a relay, then a user can access a room by making a clocking (e.g. ENTER1, 04) on the SL-870 terminal (addr. 30h).

The same SL-870 might be configured as (ENTER2,05) which would open another door next to the terminal, via AUX ID Event #3, associated to this combination.

The AUX ID Event Default serves to catch all the cases not defined in the AUX ID Event table. This should be used to set the Generate Clocking flag, so that all clockings will generate a record.



6.2. AUX Commands

Direct commands from the master PC to an AUX unit are possible using two terminal commands:

AuxTx ('x') and AuxRx ('z').
These commands are carried out immediately and have a higher priority than the cyclic AUX functions defined in the CNF8x03 program.

The mechanism is as follows:

Example:

If the user wants to send a command to an AUX unit, the command is sent to the terminal, which then sends it on to the AUX unit in the next cycle. If the command is such that it will generate a reply from the AUX unit, this reply is read from the AUX unit in the following (or one of the following, if an error occurs) cycle and is then put into the AuxCmd.buffer. This buffer is read by the master PC with an AuxRx command.

The data flow is as follows:

If we want to read EEPROM location 32h on the AUX unit with the address 21h, the master PC would send the following string to the terminal:

"21R32"

The '21' is converted into a number and placed into AuxCmd.addr
"R" and "32" are placed into the AuxCmd.buffer, beginning from location 01h. Location 00h contains the number of bytes to send (02h in this case).

Buffer[0]	Buffer[1]	Buffer[2]
02h	‘R’	32h

Note that the string is parsed so that the first two characters ‘02’ are converted to a single byte, the next character ‘R’ is copied as-is into buffer[1]. All the following characters in the string are assumed to be bytes represented as two hexadecimal ASCII characters.

After receiving and parsing the string, the AuxTx command (firmware of the SL-8x terminal) checks the LSB of the command letter (‘R’ i.e. 52h in this case) and since it’s 0, the AuxCmd.status field is filled with 3. This means that after having sent the contents of the buffer to the AUX device, the terminal expects to receive an answer in the next cycle and will try to read data from that AUX device.

The AUX device will receive the bytes in AuxCmd.buffer[0] to AuxCmd.buffer[2], recognise the ‘R’ command to mean “ReadEEPROM”, read address 32h of the on-board EEPROM and prepare the following in the transmit buffer:

TXbuf[0]	TXbuf[1]	Txbuf[2]	Txbuf[3]	TXbuf[4]
04h	‘R’	32h	12h	34h

Txbuf[0]	this is the number of bytes to be sent, not including the length byte itself
Txbuf[1]	these two bytes are copies of the original command and its parameter
Txbuf[2]	
Txbuf[3]	this is the actual data read out from the EEPROM
Txbuf[4]	(we assume that 1234h was the 16-bit value read from address 32h)

In the next cycle, the terminal will read data (Txbuf[0] to Txbuf[4]) from the AUX device, place it into the AuxCmd.buffer, set AuxCmd.status = 8 to indicate that there is data ready to be read, and wait for an AuxRx command from the PC.

6.3. Terminal commands for AUX commands

The following commands from the PC to the SL-83 concern direct commands to AUX units.

<u>AuxTx ‘x’</u>
Transmits a string to the AUX unit with the address AA. The string contains the command letter C and the data (DD) (optional).

PC → Terminal

N(AA)C(DD)...(DD)<LRC>

Where:

N: number of transmitted characters + 20h
This number includes AA (2 characters) , C (1 character) , DD (2 characters each) and LRC (1 character)

(AA): address of AUX unit
C: command
(DD) Data (0 to 18 bytes)

Terminal→PC

<ACK> ‘0’	OK
<ACK> ‘1’	AuxCmd channel busy

The terminal will send the characters to the AUX device with address (AA) in the next cycle. If the send routine is successful, the state machine needs to know if there is to be an answer from the AUX device in the form of a string that needs to be received in the following cycle.
This is achieved by defining that the last bit of the command letter is:

0 (xxxx xxx0): returned data expected (status <- 03)
1 (xxxx xxx1): no returned data expected (status <- 01)

So, if the command letter is 'odd': 'a' (71h),'c' (73h), 'e' (75h), etc. the associated command must be one that doesn't generate a reply from the AUX device, while the 'even' letters: 'b' (72h), 'd' (74h), etc. have to be associated to commands that all generate a reply from the AUX device.

<u>AuxRx 'z'</u>
Receives the contents of the Aux command buffer if it's ready (has been filled with data from an AUX unit as a result of a previous command sent to it by AuxTx).

Terminal→PC

N(AA)C(DD)...(DD)<LRC >

Where:

N: number of transmitted characters + 20h
This number includes AA (2 characters) , C (1 character) , DD (2 characters each) and LRC (1 character)
(AA): address of AUX unit
C: command
(DD) Data (0 to 18 bytes)

or
<ACK> '1' Aux command buffer not ready
<ACK> '2' N in command buffer <3 or >20

6.4. AUX MASTER firmware

The SL-83 firmware uses separate data structures for the cyclic AUX functions and the direct AUX commands.

6.4.1. Cyclic funtions

The cyclic AUX structure is as follows:

```
typedef struct
{
       unsigned char addr; // AUX address
       unsigned char func; // function, the same unit can have several
       unsigned int  counter;     // decremented at each firmware cycle, function executed
       unsigned int  MAXCNT;      // when counter reaches 0. MAXCNT = refill value
       unsigned char status;      // status value, used for state machine state
       unsigned char buffer[22];  // send or receive buffer
} AuxFunc;
```

The AuxFTab[8] is defined as type AuxFunc, so we have an array of 8 such structures for the cyclic functions.

There is also a global variable AuxInd, which is incremented at every pass through the main firmware cycle. When it reaches 8, it is reset to 0. The AuxInd is used as an index into the AuxFTab[8] array. At every pass through the firmware, the next element of the AuxFTab[8] array is indexed: its counter value is decremented and checked if it has reached 0, whereupon the function will be executed and the maximum counter value will be set from the MAXCNT field.

The addr, func and MAXCNT fields are filled in from the serial EEPROM at initialisation time. These values are set by the configuration program CNF8303.

6.4.2. Aux commands

The data structure for the Aux commands:

```
typedef struct
{
      unsigned char addr;        // address of AUX device
      unsigned char status;            // Cmd state machine state
      unsigned char buffer[22];  // Rx/Tx buffer
} AuxCmd;
```

DoAuxCmd() function

The DoAuxCmd function is a simple state machine that uses the following variables to define its function:

AuxCmd.status this defines the state of the AuxCmd state machine

The following values (state machine states) are defined:

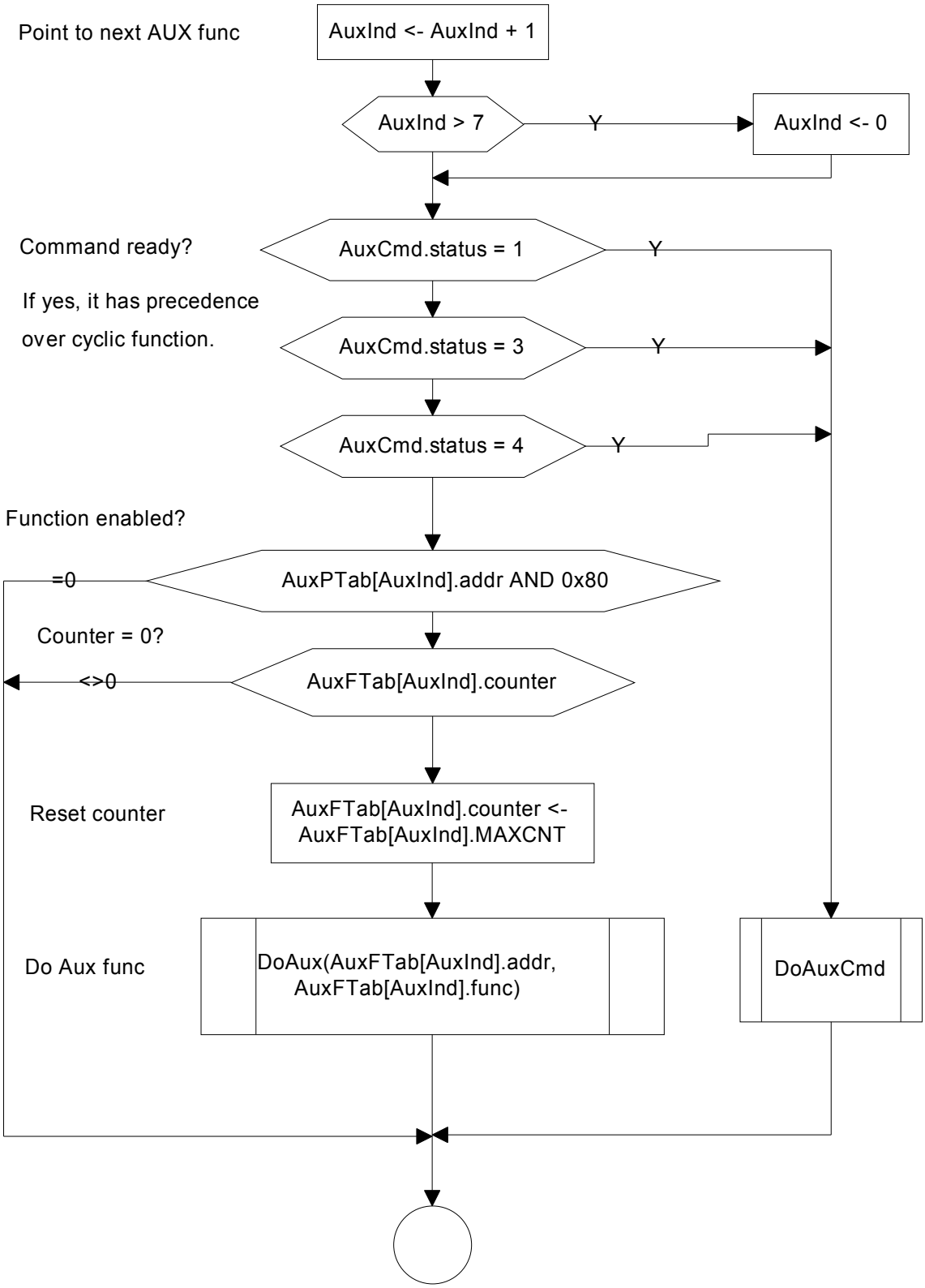
1 The array in the AuxCmd.buffer should be sent to the AUX device
No response is expected from the AUX unit

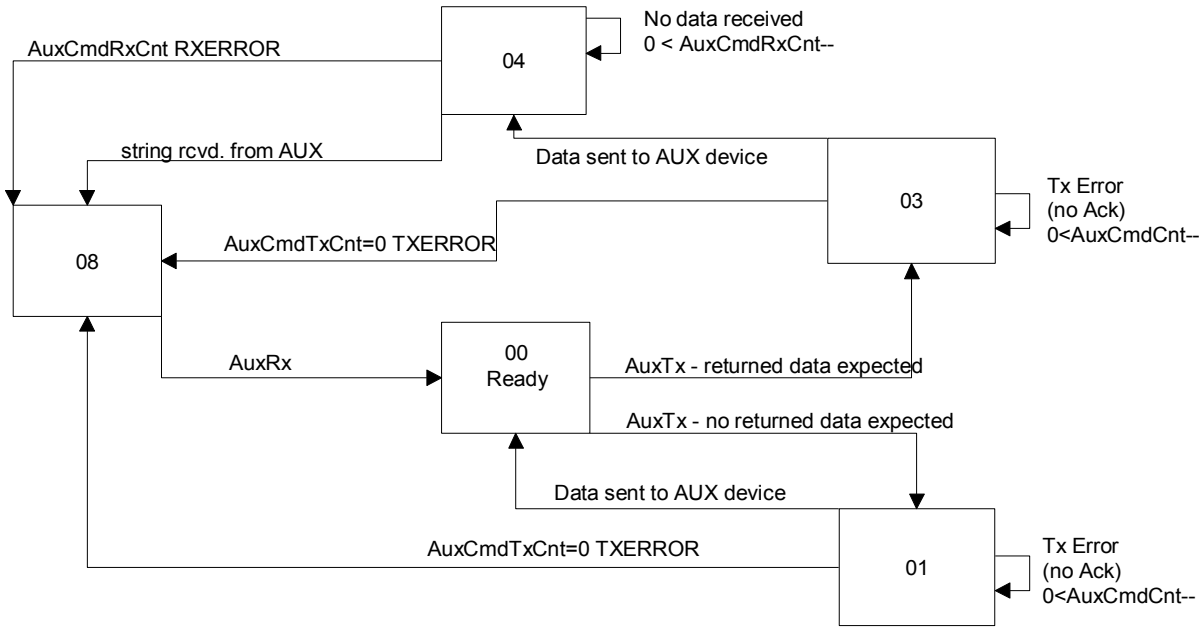
3 The array in the AuxCmd.buffer should be sent to the AUX device
A reply array is expected to come from the AUX device in the following cycle

4 The array in the AuxCmd buffer has been sent to the AUX device and a reply array is expected to come from the AUX device in the next cycle.

8 An array from an AUX unit has been received and is ready to be read by an AuxRx command.
(Or a TxError condition has happened)

This flow chart illustrates the section of the SL-85 main loop that deals with AUX cyclic functions and commands:





AuxCmdRxCnt and AuxCmdTxCnt are set in the AuxTx an AuxRx command
The lowest bit of the C(ommand) specifies whether returned data is expected or not i.e. do we go to state 3 (returned data) or state 1 (no data).
xxxx xxx0 -> status=03, returned data
xxxx xxx1 -> status=01, no returned data

The above flowchart illustrates the AUX direct command state machine, which is implemented in the DoAuxCmd function (see previous flowchart).

7. Application examples

Some of the possibilities:

1) Access control to a room with 2 entrances. Each entrance door strike is activated with its own relay on the controller board and a reader is mounted on the outside of both doors. Users exit by activating the egress buttons which are mounted on the inside of each door.
2) Access control and entry/exit recording to a room with one door. Readers are mounted both on the inside and the outside of each door. Only one relay is used.
3) Time & attendance. The controller is used for simple clocking, the relays are not used.
4) Connecting the controller to an alarm system (example for the use of conditional relay blocks)
5) Generating events at specific times and/or dates. E.g. a siren to signal the end of a shift.
6) Random events for spot searches of employees coming or going.

There are other situations that can be covered by changing the configuration parameters.

Application Examples:

We will use the previously described examples to illustrate the configuration of the terminal.

7.1. Access control, room with 2 doors

Every reader controls its own relay. Egress buttons are used for exit.

	RL1	RL2	REC
T1	-	-	NO
TA1	-	-	NO
TB1	-	-	NO
TC1	RLT	-	YES
T2	-	-	NO
TA2	-	-	NO
TB2	-	-	NO
TC2	-	RLT	YES
S1O	OFF	-	YES
S1C	-	-	YES
S2O	-	OFF	YES
S2C	-	-	YES
E1	RLT	-	YES
E2	-	RLT	YES

The time dependent action (TC) was used.

Note the setting of S1O and S2O (the door-opened events). When the door is opened, the relay is deactivated immediately. This conserves energy, which is very important when the device is working off the battery during a power cut. It should be noted that door strike solenoids can consume as much as 2A.

This scenario records all the events of access and door opening and closing but since exits are not recorded with an iButton (you can’t tell who pressed an egress switch) migration reconstruction is impossible.

7.2. Access control and entry/exit recording, room with 1 door

The door is opened with active badges both from the inside and outside. Egress buttons can also be used within a door-phone system or by a guard or secretary for granting access to visitors.

Reader 1 is mounted on door 1 which is opened by relay 1, while reader 2 is mounted door 2, which is opened by relay 2.

	RL1	RL2	REC
T1	-	-	NO
TA1	-	-	NO
TB1	-	-	NO
TC1	RLT	-	YES
T2	-	-	NO
TA2	-	-	NO
TB2	-	-	NO
TC2	RLT	-	YES
S1O	OFF	-	YES
S1C	-	-	YES
S2O	-	-	YES
S2C	-	-	YES
E1	RLT	-	YES
E2	-	-	YES

7.3. Simple working hours recording, no access tables or relays

Clockings on each of the two readers are generated with unique activity codes. These codes can later be substituted by chosen activities such as: IN, OUT, OUT-OFFICIAL, OUT-LUNCH.

	RL1	RL2	REC
T1	-	-	YES
TA1	-	-	NO
TB1	-	-	NO
TC1	-	-	NO
T2	-	-	YES
TA2	-	-	NO
TB2	-	-	NO
TC2	-	-	NO
S1O	-	-	NO
S1C	-	-	NO
S2O	-	-	NO
S2C	-	-	NO
E1	-	-	NO
E2	-	-	NO

7.4. Example for the application of the conditional relay blocks

A company uses the SL-83 and also has an alarm system. The two have to be linked so that the activation / deactivation of the alarm can be done automatically by the SL-83.

There will be three types of users:

1. ordinary employees, who can enter during specific periods of the day,
2. executives, who can enter the company when there is nobody else inside (early in the morning) and can deactivate the alarm,
3. the owner, who can also activate the alarm.

Notes:

- The owner can do all that the executives can, i.e. enter at any time of day and deactivate the alarm.
- Activation of the alarm is possible only if there is nobody left in the company.
- The alarm has a voltage-free input which is used to toggle it between active and inactive modes. Shorting these inputs for at least a few seconds toggles the system into the opposite state.
- The state of the alarm (active or inactive) can be read from a voltage-free output (open-collector).

Setup

Reader #1 will be used as the IN reader and for the deactivation of the alarm, while #2 will be used as the OUT reader and for the activation of the ALARM.

We will use the activity bits in the following way:

A – deactivation
B – activation
C – opening the door (with time discrimination)
We will be using the hardware in the following fashion:
Relay1 (RL1) is used to toggle the alarm system
Relay2 (RL2) is used to drive the door strike solenoid (opening the door)
Sensor1 (S1) is used to monitor the state of the alarm system , i.e. is it active or inactive

Access tables – Kata

Using the Kata program we can assemble and upload the access tables.

- Ordinary employees only have the C bit set. They can enter the premises but not deactivate or activate the alarm. The time discrimination (WeekType) should be set so that these employees can not enter early in the morning or during weekends.
- Executives have the C and A bits set. They can enter the premises and when they enter the alarm will be deactivated if it is currently active. The time discrimination should be set so that they can (and should) enter the premises before the ordinary employees so that they can deactivate the alarm before anybody that can't deactivate the alarm arrives.
- The owner has all three bits set: A, B and C. He can do all that the executives can and he can also activate the alarm when he leaves. His time discrimination should be set so that he can always enter the premises.

Configuration – Cnf8303

Using the Cnf8303 program, we can set the hardware configuration of the SL-83 controller.
In order to get a clearer perspective, we will only list the table elements that are used. All the others are set to the default value: "No Effect".

T1A	RL1 ON RLT sec.	IF S1=0	REG=Y	deactivation, if currently active
T1C	RL2 ON RLT sec.	IF always	REG=Y	open door
T2			REG=Y	record exit
T2B	RL1 ON RLT sec.	IF S1=1	REG=Y	activation, if currently inactive

We did not use the sensor inputs to turn the door strike relay off when the door is opened, nor did we use the egress inputs to open any doors. We assumed that the doorknob is used to exit the premises, not the relay. All this was done to keep the example simple and concentrate on the use of the "conditional block" of the configuration of relays #1 and #2.

Connecting the SL-83 to the DSC 5010 alarm system

Since the alarm needs a voltage-free contact for its arm/unarm input, we need to set the J2 jumper block to the NV/NO setting and connect the terminals as follows: