MGL ECB RS232 protocol
Document date March 20, 2015

General

This document describes the RS232 protocol used with the MGL Avionics ECB (Electronic Circuit breaker).

MGL Avionics makes this document available to interested third parties without any implied warranties or guarantees. Any third party implementation accepts all risks associated with such implementation.

Hardware

The ECB module contains one RS232 interface consisting of a RX and TX line using standard RS232 voltage level signaling.

Baudrate is 57600, 8 data bits, 1 start bit, 1 stop bit.

The protocol allows for up to 8 ECB modules to be connected to a single RS232 host port.

This is done by daisy-chaining the modules as follows:

Host TX goes to RX of first module. TX of first module goes to RX of next module and so on. The TX of the last module goes to RX of the host.

Each module needs to have a unique node address. This is set by means of dip switches 1,2 and 3 giving a total of 8 nodes which equates to 64 breakers.

Nodes can be wired in any order.

If two or more nodes share the same node address, the node that is closest in line to the RX of the host wins the slot and data from other nodes with the same address is lost.

Protocol

The host will receive a data packet every 100mS. The host may send commands.

The host may receive a setup data packet if it has requested this by a command.

Data types used:

Byte: 8 bits

Word: 16 bits

Longint: 32 bits

Multibyte items are little endian

The data packet

$D5 //Start of message

$82 //Start of message

$01 //Message type

$5C //length of data following (excludes checksum at the end of packet)

Flags: longint; //First byte used: Each bit is set by one module in the chain according to its node address. Allows the host to identify which nodes are connected.

BreakerStates: array[0..7] of byte; //One byte per module = 8 breakers.

Bit is 0 = breaker off, Bit is 1 = breaker on.

FaultStates: array[0..7] of byte; //One byte per module = 8 breakers

Bit is 0 = breaker has not faulted, Bit is 1 = breaker has faulted (tripped).

Currents: array[0..63] of word; //64 currents

Currents for each breaker in steps of 0.1A

Voltages: array[0..7] of word; //One per module

Voltages measured at power input terminal in steps of 0.1V

Doubling: array[0..7] of byte; //one byte per module

0 = No breaker doubling
1 = Breakers 1+2 joined
2 = Breakers 1+2 and 3+4 joined
3 = Breakers 1+2+3 joined
4 = Breakers 1+2+3 and 5+6 joined

Defaults: array[0..7] of byte; //one byte per module

Power on defaults for each breaker: 0 = off, 1 = on.

Options: array[0..7] of byte; //one byte per module

Bits 0,1

- 0 = Wigwag disabled
- 1 = Wigwag off
- 2 = Wigwag steady on
- 3 = Wigwag flash

Bit 2

- 1 = programmable profile is selected

Bits 3,4

- 0 = no flasher
- 1 = flasher on breaker 5
- 2 = flasher on breakers 5 and 6

Bit 5

- 0 = single LED mode, 1 = dual LED mode

WigWagTime: array[0..7] of byte; //one byte per module

in steps of 0.1 seconds (interval is this value * 2)

Profile: array[0..7] of byte; //one byte per module

Number of the selected profile (profile selected on dipswitch array, see user manual for profiles). Note: zero based. Profile 1 in manual is “0”.

Checksum: One byte. Modulo 8 addition of all bytes in data package starting with Message type.

Checkxor: One byte. XOR all of the bytes in data package starting with Message type. Exclude checksum.

The setup packet

$D5 //Start of message

$82 //Start of message

$00 //Message type

$19 //Data length

TripcurrentSet: array[0..7] of word;

Trip currents in steps of 0.1A

TripTimeSet: array[0..7] of byte;

Trip times in steps of 0.1 seconds

Checksum: One byte. Modulo 8 addition of all bytes in data package starting with Message type.

Checkxor: One byte. XOR all of the bytes in data package starting with Message type. Exclude checksum.

Commands

Commands are sent by the host to a module. The command includes the destination of the command (node address).

All commands follow a similar protocol to the packets sent by the ECB.

$D5 //Start of message

$82 //Start of message

$nn //Message type

$nn //Data length (number of bytes following excluding two check bytes)

$nn //Node address 0 to 7 (breaker groups 0-7 to 56-63)

Variable length data

Checksum: One byte. Modulo 8 addition of all bytes in data package starting with Message type.

Checkxor: One byte. XOR all of the bytes in data package starting with Message type. Exclude checksum.

Set trip current - Message type 2

$nn: Byte; //Breaker number 0-7 (within node)

$nn: Word; //Trip current in steps of 0.1A. 100 = 10A.

Non-volatile storage for this item in device.
This item is relevant and used only for the programmable profile

Set trip time - Message type 3

$nn: Byte; //Breaker number 0-7 (within node)

$nn: Byte; //Breaker trip time in steps of 0.1 second. 10 = 1 second.

Non-volatile storage for this item in device.
This item is relevant and used only for the programmable profile

Wigwag - Message type 4

$nn: Byte; //Wigwag mode. 0=off, 1=steady on, 2= flash

Wigwag time set - Message type 5

$nn: Byte; //Wigwag time in steps of 0.1 seconds (interval is this time * 2)

Non-volatile storage for this item in device.

Set breaker - Message type 6

$nn: Byte; //Breaker number 0-7 (within node)

$nn: Byte; //0 = breaker off. 1 = breaker on (also reset breaker).

Set breaker doubling - Message type 7

$nn: Byte;

0 = No breaker doubling
1 = Breakers 1+2 joined
2 = Breakers 1+2 and 3+4 joined
3 = Breakers 1+2+3 joined
4 = Breakers 1+2+3 and 5+6 joined

Non-volatile storage for this item in device.
This item is relevant and used only for the programmable profile

Wigwag takes its cue from here for breaker doubling (used if wigwag is used on breakers 1+2 and 3+4). This value is also sent as part of the status message allowing the host to know when to add currents to show a total for a group of breakers.

Set dual LED - Message type 8

$nn: Byte; //0 = Single LED. 1 = Dual LED.

Non-volatile storage for this item in device.

Wigwag enable - Message type 9

$nn: Byte; //0 = Disable Wigwag. 1 = Enable Wigwag.

Non-volatile storage for this item in device.
This item is relevant and used only for the programmable profile.

Wigwag will be enabled for breakers 1 and 2 or 1+2 and 3+4 if breaker doubling set for that combination. Any other doubling setup will disable wigwag function.

Startup default - Message type 10

$nn: Byte; 8 bits, each used to set the default state of a breaker on startup provided breaker panel switch is on. Bit 0 is for breaker 0 and bit 7 for breaker 7.
0: Breaker will start in the off state regardless of state of panel switch.
1: Breaker will start in the on state of panel switch is in the on state.

Flasher mode - Message type 11

$nn: Byte; //0 = No flasher, 1 = Flasher on breaker 5. 2 = phased flashers on breakers 5 and 6.

Non-volatile storage for this item in device.
This item is relevant and used only for the programmable profile.

Set to factory default - Message type 12

$00: Byte; //Set to 0

Sets all non-volatile items to factory defaults.

Request setup packet - Message type 13

$00: Byte; //Set to 0

Results in transmission of a setup data packet.

Latency

The daisy chain linked ECB modules are subject to latency as a data packet ripples through the chain. The data packet is just under 100 bytes long which results in a transmission time of under 20mS. Nodes receiving a data packet will synchronize their own transmissions to coincide with the end of reception of the data packet to minimize latency.

Assuming a full complement of 8 ECB modules, the first module in the chain should thus have the highest latency and it would take 8 x 20mS = 160mS for its data to arrive at the host. This is not really an issue for an ECB application but is mentioned here nevertheless.

Latency applies for messages sent by the host as well, however the commands are very short so total latency is all but negligible.

Message loss

It is not expected that there will be message loss to be dealt with in a normal installation as connections tend to be short and well made.

The host should verify execution of its commands by monitoring the status data packet which contains nearly all required information. For breaker trip currents and trip times the host should request the setup data packet after sending any commands that would change these values to verify that messages have been received and executed buy the node.

Sample application

MGL Avionics provides the source code for its RS232 ECB application (download on www.MGLAvionics.co.za).

This is written in Delphi and may be used as reference or core for a user application.