Richard's stuff

ELECTRONICS

HONEYWELL GALAXY G2 RS485 BUS

9TH MARCH 2019 | RICHARD | LEAVE A COMMENT

I always fancied adding Ethernet to my home Alarm system, even though I don't have any particular use for it. I live in a pretty safe area and it rarely has false alarms or anything else I really need to monitor. Besides, if you're at work, miles from home and in the middle of something, what what are you going to do if you get a notification that your house alarm has gone off?

Eventually, by chance, I picked up a second hand official Ethernet adapter for the alarm at a car boot sale, but it turned out the firmware was too old and the cable to update it is surprisingly expensive, and the update itself not publicly available anyway. I was already aware of the LCE-01 from SM Security Alarms as a cheaper alternative, with more functions like a Virtual Key Pad app, but £66 for a device I didn't really have a use for was still a bit steep for someone who hates to part with money. I contacted the maker to find out about updating the firmware on mine and he offered me a good deal on a part exchange so I went for it.

The LCE-01 is a custom PCB which basically combines an off-the-shelf dev board (containing Ethernet/ microcontroller/ rs485 devices) with a buck dc-dc power supply and a few connectors. Very much the kind of thing you could make yourself – indeed the FAQ on the website says this and offers the firmware if you want to. They even say if you can make them much cheaper they'll subcontract the manufacture to you. Of course you can't really make them cheaper in any quantity, the biggest cost is the dev board and by the time you've bought that, the custom PCB, the other handful of parts and accounted for the time spent soldered them all together then they clearly aren't making a big profit on them. The main reason to make your own would be flexibility, using it to connect to something other than their service. However that service costs just £1/month, including up to 20 SMS messages. Making your own device to save £1/month, by connecting to a less tried and tested platform of your own making, is a false economy. This project is just for fun, once I've finished playing with my backup alarm I'll be sticking the LCE-01 on my real panel. Although I could put my own adapter along side, in listen only mode, to capture the events and do something else with them too.

First things first, how do you interface with the alarm? That much you can get from the manual. It has an RS485 bus that keypads and the Ethernet adapter (as well as other accessories: e.g. RF interface, RIOs) connect to. You'd think someone would have already documented this protocol, but it seems not. Projects like openGalaxy talk over RS232 and just seem to process SIA messages. The most interesting thing I found was a project on GitHub called SolarSystem where the author had made his own control panel to talk to some galaxy peripherals such as the keypad and max4 rfid readers. Unfortunately he didn't docu-

ment the protocol and the code was not written in a way that looked as though it was intended to be read by anyone else.

Sniffing the RS485 bus allowed me to work out the basic protocol for the Keypad. Interfacing directly with a keypad (without the alarm) allowed me to finish this by testing out extra options not seen on the real bus. Of course there could be other commands I haven't seen but, we'll never know unless they do crop up at some point.

So first step is to document the RS485 protocol for the system generally and then keypad and Ethernet devices.

The RS485 is half-duplex and runs in master/slave mode. Connecting to it with a PC is a simple job of hooking up an RS485 to RS232/USB and using a terminal program* with settings 9600 8 N 1. The alarm control panel is the master and initiates communication by regularly polling each slave device, at which point the slave is allowed (and expected) to send back a response.

Devices have an ID, and the first byte of each message is the device ID of the target. On the G2-44 the device ID of the panel is 0x11. The last byte of the message is a checksum. The checksum algorithm, which is the only thing I got out of the SolarSystem code, is as follows:
Add 0xaa to the sum of all the bytes of the message. This will overflow a byte so needs to be done in a larger data type. Split the larger number into bytes and add them together (ignore further overflows at this stage).
e.g. for the simple two byte message "0x11 0xfe" => 0xaa + 0x11 + 0xef = 0x01b9 => 0x01 + 0xb9 = 0xba => complete message with checksum "0x11 0xfe 0xba".

Or in C:

```
byte csum;
byte len = 2;
byte data[] = {0x11, 0xfe};

int temp = 0xaa;
for (int i = 0; i < len; i++) {
  temp += data[i];
}
csum = (byte)(((temp & 0xff000000) >> 24) + ((temp & 0xff0000) >> 16) +
```

When the panel starts up it polls for all possible device IDs to see what is connected. The format of the poll varies by the device type that is allowed to use that ID but, as with all messages, starts with the target id and ends with a checksum. There is nothing stopping a single physical device replying to more than one device ID and appearing as two devices to the panel. e.g. the prox keypad responds to a keypad id and a prox reader id. Equally the LCE-01 replies as a keypad and an Ethernet device. For some devices, such as keypads, you can select the id with a rotary selector switch on the device. The poll is sent up to 4 times before the next device id is checked. I don't know how long a device is given to respond but it isn't long, only a few fractions of a second, the 4 polls appear pretty much together on my serial terminal.

Known devices are re-polled periodically. All possible devices are re-polled (in the same way as at start-up) when exiting engineer mode, and the user is notified of additions/ removals at the keypad.

Polling sequence from G2-44. Each message is sent 5 times in rapid succession if there is no response. Of note is that my panel runs firmware v1.56 and three of these polls contain that number in some form. Bytes sequence "`0x01 0x56"` appears in polls for devices 84 and 85. Byte sequence "`0x31 0x35 0x36"` (ascii for "156") in poll for device 25.

```
10 00 0E C8
20 00 0E D8
30 00 0E E8
40 00 0E F8
02 00 0E BA
03 00 0E BB
04 00 0E BC
05 00 0E BD
84 00 0E 03 00 1A 39 7A 01 56 00 65
85 00 0E 03 00 1A 39 7A 01 56 00 66
93 00 0E 4C
92 00 0E 4B
91 00 0E 4A
90 00 0E 49
31 00 0E E9
25 00 0E 0F 00 31 35 36 89
```

*I found filtering the RS485 data on the PC a bit of a pain. I've never really used serial in a proper language. I tried to whip something up in C# (which I use for most desktop development these days) but it doesn't seem to be very good at it. I needed something that was fast and that worked well, so I ended up writing a simple program for an STM32 "blue pill" to act as a filtering converter between RS485 and RS232 to relay what I wanted, nicely formatted, to a serial terminal on the PC.

ALARM G2 GALAXY HONEYWELL RS485 STM32