Richard's stuff

ELECTRONICS

HONEYWELL GALAXY KEYPAD (CP038) RS485 PROTOCOL

9TH MARCH 2019 | RICHARD | LEAVE A COMMENT

The KeyProx model contains two logical devices, the keypad itself and the RFID reader. Logically these appear to be completely independent, though a quick look in the case confirms there is only really one device as you'd expect. This means the keypad information will be the same for a non-prox model. The prox part will probably be the same for the separate max3/4 RFID reader but I don't have one of those to compare against.

The device ID is set using the rotary selection switch on the PCB. Although it has 16 positions only 0-3 are valid for keypad ID. Positions 0/1/2/3 correspond to RS485 bus IDs of 10/20/30/40 respectively for the Keypads and 90/91/92/93 for the associated prox readers. A strange pair of numeric sequences. You'll also note from the startup polling sequence in my previous post that the prox readers are polled separately from the keypads and in reverse order. I'll just deal with the keypad in this post, as I haven't played with the prox reader much.

When the keypad is powered on it will beep and flash until it starts to receive commands. Once running, if it does not receive a command within a certain amount of time (e.g. lost contact with panel) it will revert to this beeping state.

Initial device poll

```
1 -> 10 00 0E C8
2 <- 11 FF 08 00 64 28
```

Remember the first byte is the destination device (10 for the first keypad, 90 for the first prox reader, 11 for the replies back to the panel). The second byte generally appears to be a command, in this case 00 appears to be the poll command, subsequent bytes are data for the command and finally is the checksum. I don't know what the 0E in the poll, or the contents of the reply mean, but they don't appear to change so it probably doesn't matter. If an ID is allowed to be used by multiple device types something in the reply must distinguish them, but as long as we copy the above we can emulate the version of the CP038 I have on my desk. New devices are only checked for on startup and when exiting engineer mode. Although I referred to this as the 'initial' device poll known devices will get re-polled periodically, so a device needs to continue to respond to this message whenever it receives it or the panel will raise a fault.

General replies

```
<- 11 F2 AE - bad checksum
<- 11 FE BA - ok
```

The “bad checksum” error can be sent in reply to any command (so isn’t mentioned again below). The “ok” message is a general confirmation to many commands.

Ok, with key reply (and/or tamper)

```
<- 11 F4 01 B1 - ok, with button 1
<- 11 F4 41 F1 - ok, with button 1 & tamper
<- 11 F4 7F 30 - ok, with tamper only
```

The “ok, with key” message can be sent in reply to several messages, but not all (see below for which). This is sent instead of the regular ok after a button has been pressed on the keypad or the tamper switch is open. In this reply the key is indicated by the third byte. The values translate to the following keys 00-09 = buttons 0-9; 0A = button B / 0B = button A (illogical!); 0C = Ent / 0D = Esc / 0E = * / 0F = #. To indicate a tamper on the keypad device (i.e. the case has been prised open/ ripped off the wall) the third byte (key indicator) will have bit 0x40 set in combination with any outstanding button that needs to be notified (e.g. 41 for tamper & button 1). If there is no button to report, just a tamper, the value is set to 7F. Unlike a button press which is one off, the tamper state continues to be reported until the tamper switch is closed.

Activity poll

```
-> 10 19 01 D4
```

Command 19 is the activity poll and is sent regularly to the device. Valid replies include “ok” and “ok, with key”. This is the main way that the alarm panel finds out that a button has been pressed, although “ok, with key” can be sent in reply to some other commands. The meaning of 01 in the message is not know.

Screen backlight

```
-> 10 0D 01 D4
```

Command 0D controls the backlight on the keypad screen. The value 01 (shown above) turns the backlight on, 00 turns it off. This command always appears to be sent immediately after a screen update command.

Acknowledge button

```
-> 10 0B 02 C7
```

Some background explanation first: The keypad needs to know that the panel has received the key press it told it about in a previous reply. When the panel sends a command to the keypad it will either get there (intact) in which case the keypad will respond or it will be lost/ damaged and the keypad will not respond/ send back an error. In either of these scenarios the panel knows something went wrong and can resend the command. The other way around, if the keypad replies to a poll from the panel with a button press message and the message is lost how will it know it didn't make it? The panel, not receiving the reply, will send the command (probably a poll) again, but how would the keypad know the previous reply was lost and that this is a resent command (to which it should reply with the same button message as the first time) rather than a new one? So all button press messages must be acknowledged and there are two ways for the panel to do this.
If the screen is updated as a result of a button press an acknowledgement can be sent with the screen update command, but if the screen does not need to be updated then this command (0B) is sent instead to let the keypad know the button press was received. The data here (02) alternates between 00 and 02 and the keypad will keep track of which it is expecting (the same alternating value is shared by the the screen update key confirmation, not two separate alternating values). Why alternate? If the panel acknowledged a button press with this command, but the reply to the command going back from the keypad was lost the panel would send the acknowledgement command again, with the same 00/02 (as it is a resend) and the keypad would see that it was a resend, not an acknowledgement of another button press. Hopefully that makes sense! Replies to this command are "ok" and "ok, with key".

Beep mode

```
1 -> 10 0C 03 02 F0 BC
```

Command 0C sets the keypad beep mode. The 3rd byte (03 above) is the beep mode and has the values 0=off/ 1=on/ 3=intermittent. When set to intermittent mode the 4th byte (02 above) represents the beep period (in 1/10ths of a second) and the 5th byte (f0 above) the quiet period in a repeating sequence. This sequence (on for 200ms and off for 24s, an occasional little chirp) is used with the "Alert! Enter Code" message. When set to on/off mode the 4th & 5th bytes are set to 00.

Screen update

```
1 -> 10 07 01 01 07 47 61 6C 61 78 79 20 34 34 20 20 20 56 31 2E 35 02 30
```

This is the big one, because command 07's data is a sequence of sub-commands and their data. This update says "Galaxy 44 V1.5" "00:00 SAT 01 JAN" and is quite straightforward because it is a whole screen refresh. However, other commands update the existing screen contents and so the keypad must maintain a record of the screen state. I don't know whether all these commands are used by the panel, but they could be so we need to process them if we are making our own keypad. The 3rd byte (01 above) contains flags. I don't know what the bit 0x01 (set in the above example) means. If bit 0x10 is set then a previous button press message is being acknowledged and the bit 0x02 will alternate between being set or not set (see 'Acknowledge button' above). The bit 0x80 is a flag that alternates between set and unset, which the keypad needs to keep track of to identify resent vs new screen update commands.

From the 4th byte onwards (the second 01 above) we are into the sub-command/ data stream. Most bytes below 0x20 are interpreted as commands and 0x20 and above as ascii characters to display. The first command 01 sets the cursor address to 0x00, which is the address of the first character on the first line. Then we have command 07 which turns the cursor off (so no underline or box). Then we have a series of ascii characters 47…35. This sequence is not null terminated, the keypad knows it has reached the end when it gets to a low value (02) that is a command. The command 02 is another cursor position command, this time it sets it to 0x40, the first character on the second line of the display. Does that mean that a line can be longer than the 16 characters that can be displayed? Probably, haven't tested that yet… Then another sequence of ascii characters 30…4E. Finally the RS485 checksum 88, which is not part of the sub-command processing.

Full list of screen update sub-commands:

- 01 – set cursor to position 0x00 (first character of the top line)
- 02 – set cursor to position 0x40 (first character of the bottom line)
- 03 – set cursor to the position specified in the next byte (the only command which has following data)
- 04 – scroll the entire display left one character
- 05 – scroll the entire display right one character
- 06 – show cursor at it's current position – block style
- 07 – hide cursor
- 10 – show cursor at it's current position – underline style
- 11 – seems to be the same as 06???
- 14 – backspace – move the cursor left one position and erase the character there
- 15 – move the cursor left one position (does not erase)
- 16 – move the cursor right one position (does not erase)
- 17 – reset display – clear all text, set cursor position to 0x00 and reset and left/right scroll
- 18 – flash display – all text flashes on and off
- 19 – stop flashing
- 20+ – display the character with this ascii value at the current cursor position and move the cursor right one position, does not need to follow any specific command and several characters may be in a row or be interspersed between other commands

There are still a few little details to work out and I'll update this page if and when I figure them out.

ALARM G2 GALAXY HONEYWELL RS485