Some notes / things learned

- Keybus protocol is a propriatary communication protocol used by Honeywell alarm systems to communicate with keypads (such as the 6160 alphanumeric keypad).
- The connection to each keypad is 4-wire: +12v, GND, Rx, Tx
- The wiring between the alarm and keypad is a star configuration. Each keypad's wires are routed to the alarm and connected in parallel.
- The "star" wiring config means that keypads cannot transmit to the alarm whenever they want. They must only transmit when commanded by the alarm system.
- The serial communication protocol is serial 0-12V, **inverted**, 4800 baud, 8E2 (except when it's not).
- Some info on keybus I have found online indicates the protocol is 8E1, but I think it is actually 8E2 (two stop bits). Transmitting to the keypad without the delay represented by the second stop bit does not appear to work. My code adds delays where needed to implement the extra stop bit. Another approach would have been to add support for 8E2 to the ModSoftwareSerial code.
- Keypads have an address associated with them. The lowest keypad address is 16, the highest is 23. Keypad addresses will already be programmed if your keypads have been connected to an existing alarm system. You can change the keypad address by pressing and holding the 1 and 3 buttons on the keypad until the display changes to "CON ADDR=XX". Enter a new address between 16 and 23 followed by the '*' key to save the change.
- My knowledge of keybus protocol is limited to what I learned by observing my Vista-20p alarm communicating with my 6160 keypads (and Mark Kimsal's great github repository https://github.com/TANC-security/keypad-firmware)
- I used an inexpensive logic analyzer purchased off ebay with the open-source Pulseview application to observe and analyze the communication between my alarm and keypads. The trace images in this documentation were created in Pulseview.
- A resistor divider is needed when connecting the analyzer to the keybus lines to level shift the 12v signals into an acceptable voltage range for the logic analyzer. My logic analyzer wants levels for high signals between 3.2-5.5v.
- The idle state of the alarm transmit line (keypad receive) is high. The idle state of the alarm receive line is low.
- The alarm will pull the transmit line low approximately 4ms before each transmit. It will pull the transmit low for about 13ms to start a polling cycle.
- The alarm system regularly polls the keypads to see if they have any data to send. My system appears to poll the keypads about 3 times a second (although I think it could be done less often).
- The alarm system also broadcasts status messages to all keypads about once every 4 seconds. My system always sends F7 messages, and sometimes also sends F2 messages when needed (more on these messages below).
- The F7 messages include the two lines of text displayed on the keypad LCD. If needed, the alarm system may alternate between a few different F7 messages to allow the LCD text to display alternating messages.
- Most messages include a checksum. The checksum can be calculated by summing all the bytes up to the checksum byte and taking the two's compliment of the result.

Polling the keypad

The alarm system polls the connected keypads to determine which keypads have data to transmit. Keypads with nothing to transmit will not respond to polling requests. Polling cycles do not use parity. A typical polling cycle for a keypad that has data to transmit appears below:

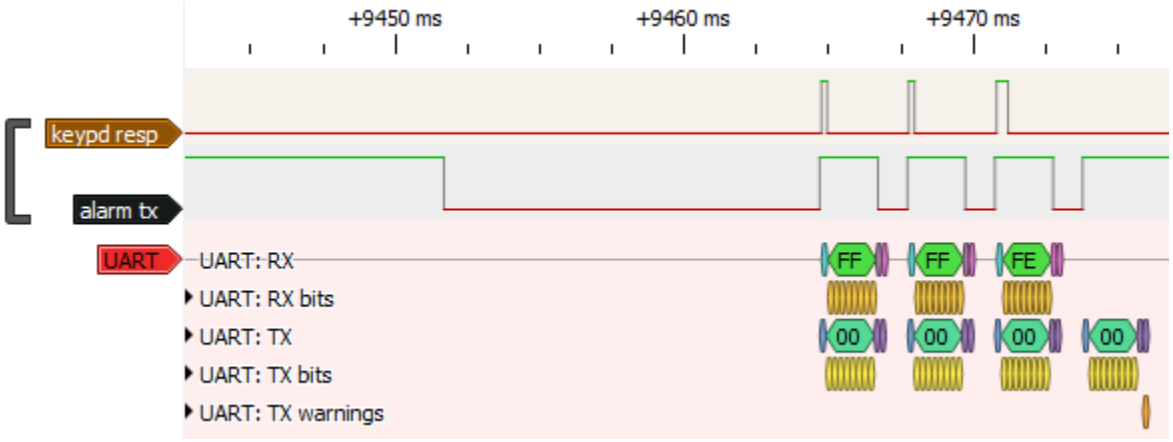

Figure 1 - Keypad Polling Cycle

The alarm starts the poll cycle by dropping the transmit line low. A low transmit of greater than 10ms is needed to start the polling cycle. The alarm then transmits a 0x00 byte three times. Each write holds the line high for 9 bit periods (start bit+byte), and is followed by delay of about 1ms.

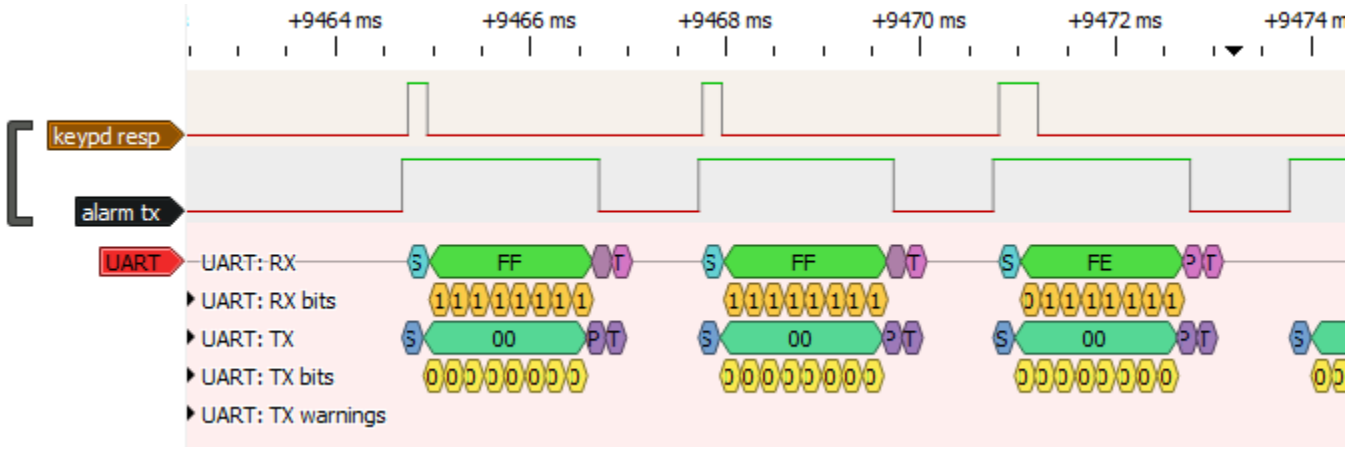

Figure 2 - Keypad Polling Cycle Expanded

If a keypad has data to send to send to the alarm, it will respond by transmitting a 0xFF (start bit high, remaining bits low) during the first two alarm transmits of 0x00. Because the keypads are using the alarm transmits to synchronize their response, multiple keypads can respond simultaneously without causing bus contention.

During the third 0x00 transmit by the alarm, each keypad transmits a byte with its own address bit low. In the trace pictured in Figure 2, the received 0xFE (lsb zero) indicates that lowest address keypad (16) is responding. If more than one keypad is responding to the polling request, more than one bit position may be low in this received byte.

After the third write of 0x00 by the alarm (and the 1ms delay), it returns the transmit line to the idle state (high).

F6 Message, Alarm -> Keypad (Fixed length, 2 bytes)

To receive data from the keypad, the alarm must command it to send the data. This is accomplished with the F6 message. The F6 message is two bytes: 0xF6 followed by the keypad address (not bit mask). The F6 message uses even parity and 2 stop bits (notice the 1 bit spacing between bytes transmitted by the alarm and keypad in the image below). A typical cycle for keypad 16 (0x10):

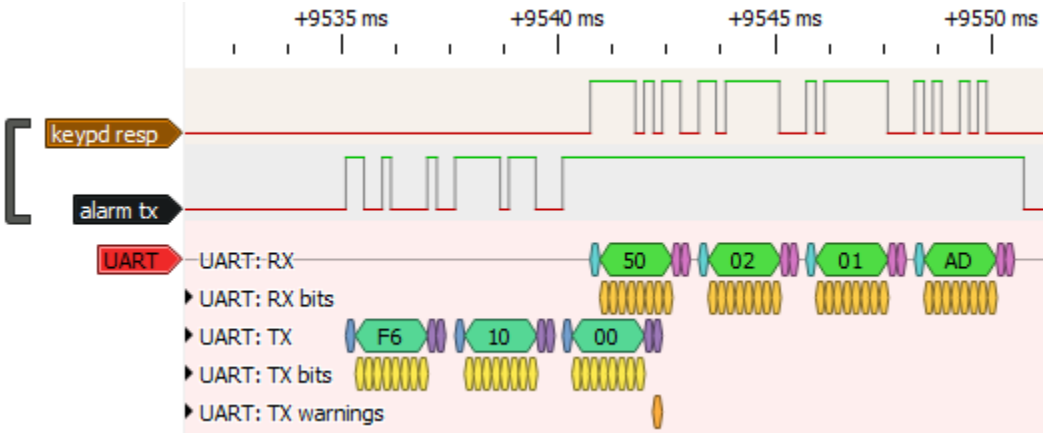

Figure 3 - F6 Message

Keypad Response Message, Keypad -> Alarm (Variable length)

The keypad responds to the F6 message (using even parity, 2 stop bits) by transmitting a message with the follow form:

Byte	Function
00	Keypad address Top two bits rotate through values 0-3 in sequence for each keypad response Bottom 6 bits are keypad address (the 0x50 shown above is keypad 0x10 (16) + sequence number 1
01	Number of bytes to follow. This will be the number of keys + checksum byte
02…	Key values: 0-9 are 0x00-0x09, 4 function keys are 0x1C-0x1F, * key is 0x0A, # key is 0x0B
Last	Checksum byte

Keypad Ack Message, Alarm -> Keypad (Fixed length, 1 byte)

The keypad will keep repeating the same output data until it has been acknowledged by the alarm. The alarm acknowledges messages from the keypad by transmitting the first byte from the keypad (keypad address and sequence number).

The full keypad response cycle is shown below.

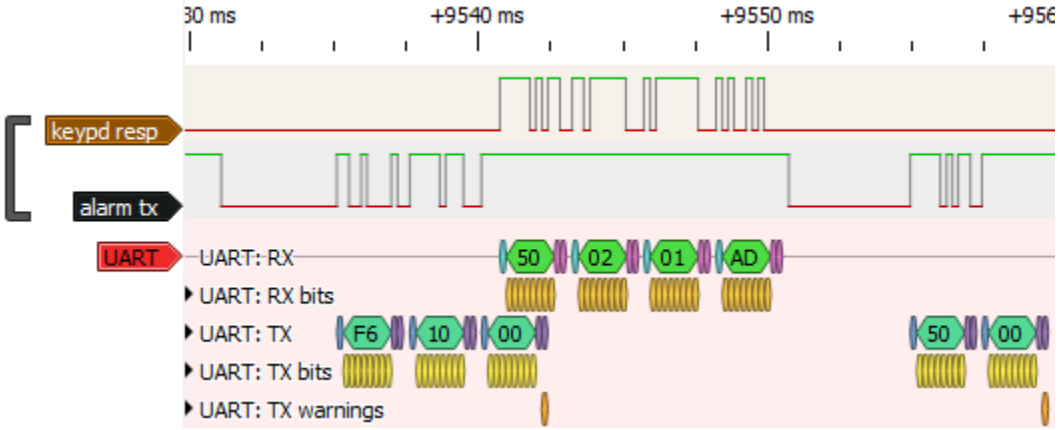

Figure 4 - Acknowledging Keypad Message

F7 Message, Alarm -> Keypad (Fixed length, 48 bytes)

The F7 message tells the keypad what to display. The size is fixed. It is transmitted with even parity and 2 stop bits. The format appears below:

Byte	Function
00	Value always 0xF7
01	Unknown (typically 0 for my alarm)
02	Unknown (typically 0 for my alarm)
03	Bitmask of keypads to receive message. lsb is keypad16, msb is keypad23. A value of 0xFF means transmit to all keypads.
04	Unknown (typically 0x10 for my alarm)
05	Zone (not sure how the keypad uses this)
06	BYTE1 – bits 0-2 control keypad tones, other bits unknown 000 – no tone 001 – chime once 010 – chime twice 011 – chime three times 100 – fast repeating tone (used when arm/disarm delay timeout is nearing completion) 101 – slow repeating tone (used for arm/disarm delay timeout) 110 – same as 101 111 – continuous loud tone (alarm triggered)
07	BYTE2 – bitmask, a high value means that setting is enabled 0x80 – Armed-stay 0x10 – Ready (if set, green Ready LED lit) Other bits unknown
08	BYTE3 – bitmask, a high value means that setting is enabled 0x20 – Chime mode enabled (not sure how this effects keypad) 0x08 – AC power is on (setting/clearing this bit does not seem to have an effect on the keypad) 0x04 – Armed-away (if set, red Armed LED lit) Other bits unknown
09	Programming mode enabled if bit 0 set. I think this is used during alarm programming. I set to zero.
10	Prompt position. Probably also used for programming, I set to zero.
11	Unknown
12	First char of line1 LCD ascii text. Msb set to 1 will turn on LCD backlight.
13-27	Remaining 15 chars of line1 LCD ascii text.
28-43	16 chars of line2 LCD ascii text
44	Message checksum
45-48	Pad bytes. Alarm always sends 48 bytes, although these last 3 appear to just be pad bytes.

F8 Message, Alarm -> Keypad (Fixed length, 9 bytes)

The function of the F8 message is unknown. The format I captured on my alarm system appears below:

Byte	Function
00	Value always 0xF8
01	0x59 on my system
02	0x10 on my system
03	0x5A on my system
04	0x03 on my system
05	0x0A on my system
06	0x0A on my system
07	0x1E on my system
08	0x10 (valid checksum for this message)

F2 Message, Alarm -> Keypad (variable length)

The F2 message passes additional info to the keypad when needed. My alarm passes this message, but I don't appear to need it to drive the keypad. **More testing needed on this message.** The format appears below:

Byte	Function
00	Value always 0xF2
01	Number of bytes (N) to follow.
02-06	Header bytes (function unknown)
07	Incremental count type (need more details)
08-18	Function unknown
19	Bit 0x02 appears to be set when alarm armed (keypad change?)
20	Bit 0x02 appears to be set if armed away is active (details needed)
21	Bit 0x02 appears to be set if in bypass mode (details needed)
22	BYTE – bitmask, a high value means that setting is enabled 0x02 – Ignore faults (exit delay) 0x04 – alarm condition (door open, etc) 0x06 – fault that does not cause alarm 0x08 – panic alarm Other bits unknown
23 to N-1	Unknown
N	Checksum

87 Message, Keypad -> Alarm (Fixed length, 9 bytes)

The 87 message is sent by the keypad once after power on. Its function is unknown. It may simply indicate to the alarm that this keypad are present in the system and provide some of its properties. This message must be acknowledged by the alarm by transmitting the first byte back to the keypad. I believe it to be fixed length because there does not appear to be a byte that indicates the length of the message. The format appears below:

Byte	Function
00	Keypad address Top two bits rotate through values 0-3 in sequence for each keypad response Bottom 6 bits are keypad address (the 0x11 shown above is keypad 0x11 (17) + sequence number 0
01	0x87 (meaning unknown)
02	0x00 (meaning unknown)
03	0x00 (meaning unknown)
04	0x04 (meaning unknown)
05	0x04 (meaning unknown)
06	0x04 (meaning unknown)
07	0x00 (meaning unknown)
08	Checksum

Here is an example trace of this message from keypad 17 (0x11):

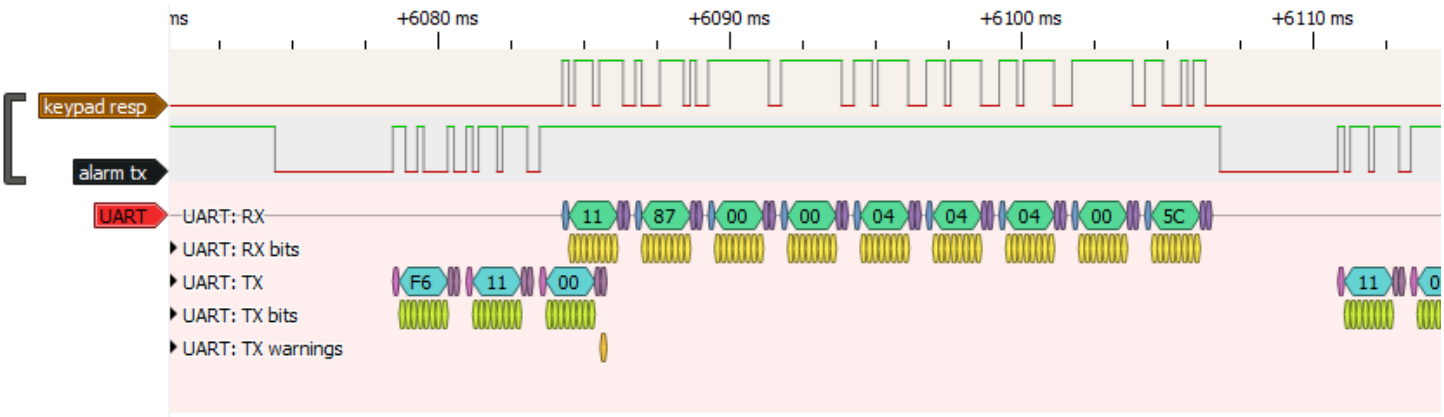

Figure 5 - 87 message from keypad