UART-Protokoll zum Extender

Durch Übermittlung durch J. Böttcher wurde der UART-Protokoll-Vorschlag von Q-loud verteilt: vgl. Informationssammlung 2020-06-03 . In der Dateiliste Docs ist das Dokument abgelegt: Ideensammlung UART Protokoll.pdf

Dieses gilt es zu prüfen und mit Q-loud zu besprechen. Agenda-Vorschlag von Q-loud:

1. Vorstellung UART-vorschläge (das bereits gesendete dokument noch mal kurz rekapituliert) - Martin Fehre
2. (Quectel) Modem Konfigurations-Optionen – Martin Fehre
3. Festlegung, welche Parameter auf dem PoC-Dashboard gezeigt werden sollen – Uwe Schnepf, bzw. all
4. Project Timeline – Stephan Keuneke

Ziel: Alle Festlegungen *__im meeting__* treffen, die notwendig sind, um den PoC abzuarbeiten.

Protokoll

Alle Angaben von Zahlen werden in Hexadecimal (0xA1 = A1h) geschrieben, außer es wird explizit als Decimal (1234d) oder Binär (1010001b) gekennzeichnet.

```
          <---HEADER---------------------------> | <-PAYLOAD-
DATA-----------------> | <CRC>
BYTE(dec): 00  01  02  03  04  05 06 07 08 09  10 |   11   12
..                  LL+11 | LL+12 LL+13
LEN (dec):                                       |   01   02
..                     LL |
Data(hex): 68 LLH LLL VER ADR OCD <SEQ 4Bytes> 69 | CMDH CHDL PL
<BLOCK1> <CMD> PL <BLOCK2> | CRCL  CRCH
```

- Header
  - FLAG_HDR_START: 0x68
  - LLH & LLL - Länge
    - 2 Bytes Länge für Payload, ohne 2 Bytes für CRC
    - WB: {0 .. 0xFFF2} = 65535 - 13 Bytes (für Hdr + CRC)
    - LLH - High-Byte zuerst (Length MSB),
    - LLL - Low-Byte (Length LSB)
  - VER - 1 Byte Version byte
    - WB: {0 .. 255}
    - Protokollversion bestehend auf einer HW- & FW- & UART-Prot.version
  - ADR - 1 Byte Adressbyte,
    - 0x0 IoT-Board,
    - 0x1 Chip Carrier Board,
    - Info: 0x2 .. 0xFF vorerst ungenutzt
  - OCD - 1 Byte Opcode
    - Config (Unidirectional)
      - 0x00 GET
      - 0x01 MGET
      - 0x02 SET
      - 0x03 MSET
    - Data (in Richtung Cloud, Bi-directional)
      - 0x06 DAT
    - 0x04 ACK (pos. Response) mit Daten **oder** ohne Ok / Done
    - 0x05 NAK (neg. Resonse) mit Daten **oder** ohne Ok / Done
  - SEQ: 4 Bytes fortlaufende Sequenznummer, die in der Antwort bestätigt wird
    - Ansteigende Nummer mit Überlauf, d.h. 0xFFFE, 0xFFFF, 0x0000, 0x0001
  - FLAG_HDR_END: 0x69
- Payload-Data für Kommandos oder Topics

- OCD=GET (1 CMD)
  - 2 Byte mit Aufbau: <CMD-ID> <CL>
  - Beispiel:
    - CMD_STATUS_RF = 0x06
    - CL = 1 (Command len)
- OCD=MGET (x CMDs - Multicommands)
  - Beispiel:
    - CMD
    - CL = 26
    - Block = string
- OCD=SET (1 CMD)
  - 3 Byte mit Aufbau: <CMD-ID><CL> <VALUE>
  - Beispiel: ein Wert
    - CMD_ENABLE_RF = 0x04
    - CL = 1 (Command len)
    - Value = 0x01
- OCD=MSET (x CMDs - Multicommands)
  - 3 Byte mit Aufbau: <CMD-ID><CL><BLOCK>
  - Block-Länge: max. 255 Bytes
  - Beispiel: ein Block mit Werten
    - CMD_CONF_HOST = 0x13
    - CL = 26 Zeichen (Command len), bei String mit '\0' (Endezeichen)
    - Block = “https://www.vsm-cloud.com“
- OCD=ACK (Response)
  - immer ACK auf CMD mit Timeout
  - 3 Byte mit Aufbau: <CMD-ID> <CL> <VALUE/BLOCK>
  - Beispiel 1: ein GET-Wert
    - CMD_STATUS_RF = 0x06
    - CL = 1 (Command len)
    - Value = 0x0 (=disable)
  - Beispiel 2: ein GET mit Block
    - CMD_CONF_HOST = 0x13
    - CL = 26 Zeichen
    - Bock (String) = “https://www.vsm-cloud.com”
- mehrfache Kommandos in einem Payload-Datenblock sind möglich, d.h. Summe der Kommandos mit Länge und Blöcken ist gleich der Länge aus Header

- OCD=NAK (neg. Response)
  - nur Sequenznummer ohne Daten
- OCD=DAT
  - Aufbau: <CMD-ID><CL><BLOCK>
  - 2 Byte
  - CMDH - MSB CMD
  - CMHL - LSB CMD
- CRC - Checksume (16bit) über Header + Payload-Hdr + Payload-Data
  - Achtung: spezielle Anordnung: zuerst LSB, dann MSB des CRC
  - CRCL - CRC LSB / Lowbyte des 16bit CRC
  - CRCH - CRC MSB / Highbyte des 16bit CRC

Ablaufsteuerung

- SRC: GET DEST: ACK (mit GET Inhalten)
- SRC: SET DEST: ACK nur Sequenznummer ohne Inhalt Ok / Done
- SRC: SET DEST: ACK mit Sequenznummer mit Inhalt NAK
  - Bsp: Set(Level=-90dB) ACK (Level=-86dB)
- Immer Handshake
- Beide Teinehmener können Aktionen auslösen

Reihenfolge der Bits und Bytes

- Bytes (Oktette) werden im Little-Endian-Format übertragen, d.h. mit niederstwertigen Bit voran, d.h.
  0xAB = 1010 1011b Übertragung: 1101 0101b
- Felder bestehend auf mehreren Bytes werden grundsätzlich mit dem höchstwertigen Oktett voran übertragen, d.h. 0xABCDEF
  Übertragung: 0xAB 0xCD 0xEF

CRC-Berechnungsforschrift

```
/************************************************************************
**
 * Function: crc_calc()
 * Purpose: generate crc value
 * Input: request or response data, byte length
 * Return: 16bit CRC
************************************************************************
**/
unsigned int crc_calc(char *buf, unsigned int data_byte_len) {
    unsigned int crc = 0xFFFF;
    for (int pos = 0; pos < data_byte_len; pos++) {
        crc ^= (unsigned int)buf[pos];      /* xor byte into least sig.
byte of crc */
            for (int i = 8; i != 0; i--) {      /* loop over each bit */
                if ((crc & 0x0001) != 0) {      /* if the lsb is set */
                    crc >>= 1;                  /* shift right and xor
0xA001 */
                    crc ^= 0xA001;
                } else {                        /* else lsb is not set
*/
                    crc >>= 1;                        /* just shift
right    */
            }
        }
    }
    return crc;
}
```

Anwendungsbeispiel:

```
/* build response crc */
unsigned int crc = crc_calc((char*) &buffer, data_bytes);

buffer[data_bytes] = (crc & 0xff);          /* LSB of CRC first */
data_bytes++;
buffer[data_bytes] = ((crc & 0xff00) >> 8);  /* MSB of CRC second */
data_bytes++;
```

Konfiguration

- Versionsnummern
  - Versionsnummer für Konfig für Protokoll
  - Idee: HW & FW-Version hier einarbeiten?

Version hier für Protokoll und/oder HW- und FW-Version (oder 2 Bytes für Vers)

- Carrier Board - Extender
  - PowerMode:
    - INT[1]
    - 0x0 - Off - Keine Funkverbindung
    - 0x1 - Standby - Funkverbindung mit minimalem Stromverbrauch
    - 0x2 - On - Aktive Funkverbindung
  - APN - SIM-Karte
    - Strings[100]
  - Benutzername & Passwort
    - String[100]
  - Signalqualität (Quectel-basierend)
    - int[1]
    - WB: (-1)*{113..0}
    - Leistungswert in dBm
    - offline 0xFF = 255 dB
  - MQTT-Server - Bestandteil des PoC
    - Typ: String[100],
    - Hostname oder IP-Adr.
    - Domain
  - Andere Server
    - anderes Protokolle
    - REST-API
    - GraphQL-API (ähnlich REST-API)
  - Zertifikat:
    - Typ: String[2048]
    - von Verschlüsselung abhängig
  - Cell-ID-Scan (optional)
    - Gewinnung von Positionsdaten
  - Fehlercode
    - “Wie geht es dir?”
    - Festlegung: Alle Fehlerbits bleieben erhalten, bis sie nach CLR-Befehl gelöscht werden. (Löschen erst durch IoTBoard angewiesen CLR Befehl)
    - int[4]
    - 0x0 - OK
    - Bitpos. Fehlercode
    - BOR-Reset: Mikrocontroller Feature aufgrund Spannungsvergung

Größe ist zu definieren / prüfen

- Zieladresse: <Hostname> oder <IP-Adresse>
  - Bsp1: https://www.vsm-cloud.com
  - Bsp2: 134.119.53.40
- String-Command (bspw. REST-API-Pfad)
  - Bsp: /api/command1

Statusabfragen

- Systemstatus Chip Carrier Board
  - Signalstärke,
  - Verbindungszustand,
  - Zelleninformationen,
  - Fehlercode,

    - ...
- Systemstatus Cloud-Verbindung
    - …

Daten für das PoC Dashboard

- Bilder:
    - IMG mit max. Größe für Logo
    - fest eingearbeitet
- Zahlen für Übertragung zur Cloud
    - #309 - ActualSpd - aktuelle Drehzahl [Hz], Ganzzahlig, {min,max}={0,999999}
    - #310 - DrvCurrent - Antriebsstrom [A], Float, {min,max}={09999.99}
    - #311 - OpHrsPump - Betriebsstunden Pumpe [h], Ganzzahlig, {min,max}={0,65535}
    - #314 - OpHrElec - Betriebsstunden [h], Ganzzahlig, {min,max}={0,999999}
    - #326 - TempElec - Temperatur Elektronik [°C], Ganzzahlig, {min,max}={0,999999}
- Demo
    - User-Param1: Pumpe ein/aus, Ganzzahlig, {min,max}={0,1}
    - User-Param2: LED ein/aus, Ganzzahlig, {min,max}={0,255}
- Config
    - Status Carrier Board: An/Aus/Standby, Ganzzahlig, {min,max}={0,2}