Softwareschnittstelle

FunkPi

Version 1.0.0

Stand: 13.02.2017

Autor: Sara Farina Fernandez

Es gibt mehrere Abschnitte für das Kommunikationsprotokoll mit dem FunkPi (unter Verwendung der Ethernet-, USB- oder Raspberry-UART-Schnittstellen):

1. SYSTEM

Die folgenden Befehle sind Systembefehle

- **HELP**
  Gibt eine Liste mit allen verfügbaren Befehlen zurück. Beispiel für Antwort nach dem Senden
  `HELP`
  ```
  200 OK
  HELP
  GETCONFIG
  SETCONFIG
  .
  ```

- **GETCONFIG**
  Gibt die Zeile config.txt Zeile für Zeile, aktuelle Netzwerk Werte (Adresse und Port), die MAC-Adresse und die Firmware-Version des Gerätes zurück. Beispiel für die Antwort nach dem Senden
  `GETCONFIG`
  ```
  200 OK
  address=dhcp
  port=6000
  %Read only variables
  actualaddress=192.168.1.110
  actualport=6000
  MACaddress=38b8eb100000
  Firmware=v1.0.0
  .
  ```

- **SETCONFIG parameter=value**
  Schreibt den `parameter=value` in die Datei config.txt. Die erlaubten `parameter` lauten:
  - `address`
  - `port`
  - `netmask`
  - `gateway`
  - `hostaddress`
  - `hostport`

  Ist der Parameter ungültig, wird ein Fehler mit der Beschreibung gesendet.
  Zum Beispiel, nachdem wir senden:
  ```
  SETCONFIG address=dhcp
  ```
  Die Antwort die wir bekommen ist:
  ```
  200 OK
  .
  ```

  Wenn der geschriebene Prozess in der config.txt erfolgreich ist

- **GETTIME**
  Gibt den aktuellen Zeitstempel in den Mikrocontroller im String Format zurück: yyyy-MM-dd HH:mm. Ein Beispiel der Antwort nach dem Senden von
  `GETTIME`
  ```
  200 OK
  2016-01-25 12:21
  .
  ```

- **SETTIME epoch_s**
  Setzt die RTC des RTOS-Betriebssystems. Es funktioniert wie Linux-Uhren, mit der Sekunden-Epoche. Daher ist der Wert, wenn wir den Mikrocontroller starten, 0 = 1970-01-01 00:00:00. Der Parameter `epoch_s` ist die Zeit in Sekunden seit 1970-01-01 00:00:00, um das richtige Datum und die richtige Uhrzeit einzustellen. Beispiel für eine Antwort nach dem Senden:
  ```
  SETTIME 1459765579
  ```
  ist:
  ```
  200 OK
  .
  ```
  Und wenn wir danach `GETTIME` senden:
  ```
  200 OK
  2016-01-25 12:21
  .
  ```

- **TESTWATCHDOG**
  Testet den Watchdog-timer. Das heißt, nach 3 bis 6 Sekunden wird das System neu gestartet.

- **REBOOT**
  Macht einen kalten Neustart des Systems.

- **EXIT**
  Schließt die TCP Verbindung mit dem Client der den Befehl ausgeführt hat. Es ist nur in der TCP-Verbindung verfügbar, nicht in den USB oder UART.

2. DISPLAY

- **CLEAR**
  Löscht den kompletten Bildschirm (LCD-Feld) mit der tatsächlichen Hintergrundfarbe. Nach dem Senden des Befehls CLEAR erhalten wir die Antwort:
  ```
  200 OK
  .
  ```

- **SETROTATION rotationValue**
  Setzt die Ausrichtung (Rotation) des Bildschirms. Der Parameter rotationValue kann die Werte 0, 1, 2, 3, haben – Wobei jeder von ihnen einer 90° Drehung entspricht. Das heißt: Ist der rotationValue=1 ist die Drehung 90°, 2 = 180° und 3 = 270°. Ist der Wert 0, ist die Ausrichtung horizontal mit der Pixelkoordinate (0 | 0) in der linken oberen Ecke des LCD-Feldes.
  Nach dem Senden des Befehls, zum Beispiel:
  ```
  SETROTATION 2
  ```
  Der Bildschirm wird um 180° gedreht und die zurückgegebene Antwort ist:
  ```
  200 OK
  .
  ```

- **SETBKCOLOR color**
  Stellt die Hintergrundfarbe des LCD-Feldes ein. Es gilt nicht für alte Zustände des Bildschirms, es betrifft nur die zukünftigen Befehle: Das heißt es ändert nicht die Farbe die auf dem Bildschirm zu sehen ist, aber senden wir den Befehl CLEAR wird der Bildschirm mit der neuen Farbe „gereinigt“. Die Parameterfarbe ist ein 16-Bit_Integer (die LCD-Farben haben eine Auflösung von 16 Bit), daher werden alle Werte über 65535 als ungültige Parameter betrachtet. Die 10 Bit Farben sind definiert als 5 Bits Rot, 6 Bits Grün, 5 Bits Blau.
  Für Blau als Hintergrundfarbe, das binäre Wert ist 0000 0000 0001 1111, in dezimal 31:
  ```
  SETBKCOLOR 31
  ```
  Der Bildschirm behält die vorherige Farbe und die zurückgelieferte Antwort ist:

```
200 OK
.
```

Senden wir nun den Befehl `CLEAR`, wird der Bildschirm blau.

- **SETTEXTCOLOR color**
  Ähnlich wie SETBKCOLOR, stellt die Textfarbe des LCD-Feldes ein. Für Rot als Textfarbe, ist der binäre Wert 1111 1000 0000 0000, in dezimal 63488:
  ```
  SETTEXTCOLOR 63488
  ```
  Die Textfarbe ist nun auf diese Farbe eingestellt und die zurückgelieferte Antwort ist:
  ```
  200 OK
  .
  ```
  Senden wir nun den Befehl `SETTEXT`, wird die Textfarbe Rot.

- **SETTEXT x y text**
  Schreibt den angegebenen Text mit der Textgröße in die Koordinaten (x | y) des LCD-Feldes. Der erste Parameter x ist der horizontale Pixel, wo der Text beginnt, y die vertikale Koordinate und „text“ der zu schreibende String. Wollen wir also den Text „Hello World!“ in die Koordinaten (50 | 40) des Bildschirms schreiben, lautet der Befehl:
  ```
  SETTEXT 50 40 Hello World!
  ```
  Der Text erscheint auf dem Bildschirm und die zurückgelieferte Antwort ist:
  ```
  200 OK
  .
  ```

  Die maximale Textlänge sind 50 Zeichen.

- **CLEARTEXT x y textLength**
  Es löscht den Bildschirm in dem LCD-Feld mit der eingestellten Hintergrundfarbe, mit dem angegebenen Pixel (x | y) und der Textlänge. Zum Beispiel, wenn bei (10 | 10) „Hello World!“ geschrieben ist und wir senden den Befehl:
  ```
  CLEARTEXT 50 40 4
  ```
  Ist die zurückgelieferte Antwort:
  ```
  200 OK
  .
  ```
  Der Bildschirm zeigt nur noch “o World!“ und an den Plätzen der ersten 4 Buchstaben ist die eingestellte Hintergrundfarbe zu sehen.

- **DISPLAYONOFF onOff**
  Schaltet den LCD_Bildschirm an und aus. Ist die Variable onOff=0, schaltet es den Bildschirm aus, ist onOff=1, schaltet sie ihn ein. Senden wir:
  ```
  DISPLAYONOFF 0
  ```
  Wird das Display ausgeschaltet und die Antwort ist
  ```
  200 OK
  .
  ```

- **SETIMAGE x y filename**
  Zeigt das Bild von der Bilddatei filename in die Koordinaten (x | y) des LCD-Feldes. Die Bilddatei muss im BMP-Format und auf der SD-Karte gespeichert sein. Senden wir
  ```
  SETFILE 0 10 myimage.bmp
  ```
  wird das Bild von myimage.bmp in die Koordinaten (0 | 10) gezeigt und wir bekommen die Antwort:
  ```
  200 OK
  .
  ```
  Die Bilddatei kann mit SETFILE hochgeladen werden (sieh SD Files Sektion).

3. SD FILES

Die maximale Länge des Dateinamens sind 8 Zeichen und die Erweiterung 3 Zeichen, also 12 Zeichen; für alle gespeicherten Dateien auf der SD-Karte. Der Dateiname unterstützt nur ASCII – Zeichen: „a-z“, „A-Z“ und „0-9“.
Zum Beispiel ist „test.txt“ gültig (4 ASCII Zeichen + Punkt + 3 ASCII Zeichen Erweiterung).

- **SETFILE fileLength filename**
  Schreibt die Datei filename mit Länge fileLength auf die SD-Karte. Nach dem Befehl, wird der Inhalt der Datei gesendet (in Bytes), gefolgt von einem Zeilenende (CR-LF). Gibt die Anzahl der geschriebenen Bytes zurück. Senden wir:
  ```
  SETFILE 25 test.txt
  ```
  wartet auf 25 Bytes+CRLF und schreibt sie in die Datei test.txt auf die SD-Karte. Wir bekommen die Antwort:
  ```
  200 OK
  25
  .
  ```

- **GETFILE filename**
  Gibt die Länge des Dateinamens der Datei von der SD-Karte, gefolgt von einem Zeilenende (CR-LF) und die Daten in der Datei als binären Inhalt zurück. Wenn wir den Inhalt der Datei test.txt von der SD-Karte erhalten möchten, senden wir:
  ```
  GETFILE test.txt
  ```
  Und wir bekommen die Antwort:
  ```
  200 OK
  25
  Content of the test file.
  .
  ```

- **DELETEFILE filename**
  Löscht die Datei filename von der SD-Karte. Existiert diese Datei nicht, passiert nichts (es wird kein Fehler zurückgegeben). Wollen wir die Datei test.txt löschen, senden wir:
  ```
  DELETEFILE test.txt
  ```
  Und wir bekommen die Antwort:
  ```
  200 OK
  .
  ```

4. GPIO AND ANALOG

- **GETANALOG**
  Gibt die Werte der 8 Analog Eingängen zurück. Es hat eine 12 Bit Auflösung, damit gibt es für jeden Eingang einen Wert zwischen 0 und 4095. Ein Beispiel für die Antworten nach dem Senden GETANALOG:
  ```
  200 OK
  28
  0
  458
  3569
  852
  10
  4000
  25
  .
  ```

- **SETGPIO gpioNum gpioValue**
  Setzt den Wert in den Schreib-GPIO-Pin. Die in gpioNum angegebene Zahl mit dem Wert in gpioValue. gpioNum muss einen Wert zwischen 0 und 7 haben und gpioValue 0 oder 1, andernfalls ist die Rückantwort „ungültiger Parameter“. Ein Beispiel der Antworten:

  ```
  SETGPIO 4 0
  ```

  ist:

  ```
  200 OK
  .
  ```

- **GETGPIO gpioNum**
  Gibt den Wert des angegebenen GPIO-Pins gpioNum zurück. Der zurückgegebene Wert ist 0 oder 1. Der gpioNum muss einen Wert zwischen 0 und 7 haben, sonst wird eine Fehlermeldung mit „ungültige Paramter“ angezeigt. Ein Beispiel für das Senden:

  ```
  GETGPIO
  ```

  ist:

  ```
  200 OK
  1
  .
  ```

- **SETLED ledValue**
  Schaltet die On-Board LED an und aus. Ist der Parameter ledValue 0, schaltet er es aus und bei ledValue 1, schaltet er sie ein. Senden wir:

  ```
  SETLED 1
  ```

  Schaltet es die LED ein und antwortet:

  ```
  200 OK
  .
  ```

5. TRANSCEIVERS

Die Transceiver haben eine Standardkonfiguration, die nach dem Start des BIOS in die Initialisierung geladen wird. Diese Konfiguration wird in 2 Dateien auf der SD-Karte gespeichert: trx433.txt für den Transceiver mit 433MHz und trx868.txt für denjenigen mit 868MHz. In diesen Dateien enthält jede Zeile eine Registeradresse und einen Wert für den CC1101, die beide in Hexadezimalzahlen ausgedrückt sind. Beispielsweise könnte der Inhalt von trx433.txt sein:

0002 06
0004 A4
002D 85

Das Register 0x02 (IOCFG0) hat den Wert 0x06, das Register 0x04 (SYNC1) den Wert 0xA4 und das Register 0x2D (FSCAL1) den Wert 0x85; Der CC1101 arbeitet bei 433MHz.

- **SETREGTRX433 regAddress regValue**
  Es schreibt auf das ausgewählte Register von dem CC1101 Transceiver, der bei 433 MHz arbeitet. Der Parameter redAddress ist die Adresse des Registers und regValue ist der neue Wert. Der gültige Wertebereich für redAddress geht von 0x00 bis 0x2E(Konfigurationsregister) sowie 0x3E (PATABLW für die Leistungskonfiguration) und 0x3F (TX FIFO), die alle schreibgeschützten Registern entsprechen. Der gültige Bereich für regValue ist 0 bis 0xFF, da die Register alle 1 Byte lang sind. Beide Parameter regAddress und regValue sind in hexadezimaler Form ausgedrückt.
  Die Antwort aus diesem Befehlt ist das Statusbyte der Transceiver. Zum Beispiel:

  ```
  SETREGTRX433 05 AA
  ```

  Es schreibt den Wert 0xAA in das Register 0x05 (SYNC0) und antwortet (vorausgesetzt, der Transceiver befindet sich im IDLE-Zustand):

  ```
  200 OK
  0F
  .
  ```

- **GETREGTRX433 regAddress**
  Liest den Wert eines angegebenen Registers von dem CC1101 Transceiver, der bei 433 MHz arbeitet. Der Parameter redAdd ist die Adresse des Registers. Der gültige Wertebereich von regAdd geht von 0x00 bis 0x3F (Konfigurationsregister, Statusregister, Multi-Byte-Register), die alle lesbar sind. Beachten Sie hierbei, dass die Befehls-Strobes, wie die Statusregister, Adressen zwischen 0x30 und 0x3D haben. Um diese letzten zu lesen muss der Burst-Bit aktiviert sein. Da diese API einen bestimmten Befehl zum Senden der Befehls-Strobes enthält, wir bei der Verwendung von GETREGTRX433 mit RegAdd-Werten zwischen 0x30 und 0x3D der Rückgabewert aus den Statusregistern (PARTNUM, VERSION, FREQEST, RXBYTES, RSS …) ermittelt.
  Der Parameter regAdd wird in Hexadezimal ausgedrückt.
  Die Antwort des Befehls ist der Wert des Registers, ebenfalls in Hexadezimal ausgedrückt. Zum Beispiel:
  ```
  GETREGTRX433 05
  ```
  Es liest den Wert vom Register 0x05 (SYNC0) und antwortet (vorausgesetzt, dass das Register 0xAA enthält):
  ```
  200 OK
  AA
  .
  ```

- **SENDCMDTRX433 cmdStrobe**
  Sendet den angegebenen Befehls-Strobe zu dem CC1101 Transceiver, der bei 433 MHz arbeitet. Der Parameter cmdStrobe ist der jeweilige Befehls-Strobe. Sein gültiger Wertebereich ist von 0x30 bis 0x3D (SRES bis SNOP). Der Wert von cmdStrobe wird in Hexadezimal ausgedrückt.
  Die Antwort des Befehls ist das Statusbyte des Transceivers, ebenfalls in Hexadezimal ausgedrückt. Zum Beispiel:
  ```
  SENDCMDTRX433 3D
  ```
  Das heißt SNOP (macht nichts) Befehl, die Anwort ist (vorausgesetzt der Transceiver befindet sich im IDLE-Zustand):
  ```
  200 OK
  0F
  .
  ```

- **SETREGTRX868 regAddress regValue**
  Es hat dieselbe Funktionalität wie `SETREGTRX433` für den Transceiver bei 868MHz

- **GETREGTRX868 regAddress**
  Es hat dieselbe Funktionalität wie `GETREGTRX433` für den Transceiver bei 868MHz

- **SENDCMDTRX868 cmdStrobe**
  Es hat dieselbe Funktionalität wie `SENDCMDTRX433` für den Transceiver bei 868MHz

- **POWERSOCKET socketCode channelCode onOff**
  Es schaltet den Kanal in der angegebenen Funksteckdose ein und aus. Es nutzt den Tranceiver bei 433 MHz (da das Wireless-Power-Socket-Protokoll bei dieser Frequenz arbeitet) und sendet das Signal um die Steckdose ein- und auszuschalten. socketCode ist der Code für die jeweiligen Steckdosen (immer mit der Länge 5); channelCode ist der Kanal in der Steckdose (A, B, C, D, oder E); und onOff hat den Wert 0 zum Ausschalten oder 1 zum Einschalten.
  Um beispielsweise die Steckdose mit dem Code „11111“ im Kanal A einzuschalten, senden wir:
  ```
  POWERSOCKET 11111 A 1
  ```
  Die Steckdose ist dann eingeschaltet und die Antwort ist:
  ```
  200 OK
  0F
  .
  ```

Wenn ein Fehler mit dem Befehl oder seinen Parametern auftritt, gibt er den Code 400 zurück, gefolgt von einer Fehlerbeschreibung. Wenn wir zum Beispiel den Server SETDATA senden, erhalten wir:

```
400 Unknown command
```

Oder wenn wir falsche Parameter wie `SETCONFIG network=123` senden, erhalten wir:

```
400 Invalid parameters
```