Lazurite Pi Gatewayのruby\SerialMonitor.rbのコードについて説明します。

Rubyのコードであることの宣言

Time out待ち時間を設定

受信したデータを取得

受信していなかったら終了

ドライバの仕様

2バイトのreadを行うと、受信したデータ長を返します。

#! /usr/bin/ruby

# -\*- coding: utf-8; mode: ruby -\*-

# Function:

# Lazurite Sub-GHz/Lazurite Pi Gateway Sample program

# SerialMonitor.rb

require 'net/http'

require 'date'

##

# BP3596 API

##

class BP3596PipeApi

##

# func : Read the data from the receiving pipe

# input : Receive pipe fp

# return : Receive data

##

def read\_device(fp)

# Data reception wait (timeout = 100ms)

ret = select([fp], nil, nil, 0.1)

# Reads the size of the received data

len = fp.read(2)

if ((len == "") || (len == nil)) then # read result is empty

return -1

end

size = len.unpack("S\*")[0]

# The received data is read

recv\_buf = fp.read(size)

if ((recv\_buf == "") || (recv\_buf == nil)) then # read result is empty

return -1

end

return recv\_buf

end

def print\_raw(raw)

fc = raw.unpack("H\*")

len = raw.length

print(len,"bytes: ",fc,"\r\n")

end

本プログラムが未対応なフォーマットを受信した時の処理。

MACヘッダを含む全データを表示します。

Payload値を文字列として認識しています。

送信先(Raspberry Pi側)のPANID,アドレス,

送信元のアドレスが格納されています。

3バイト目のデータが省略されていますが、ここにはシーケンス番号が格納されています。ドライバ内で使用していますが、Ruby側では不要なデータです。

“21a8”はIEEE802.15.4eのMACヘッダです。

Lazurite Sub-GHzから送信したデータは、この値になります。

def print\_raw2(raw)

len = raw.length

header = raw.unpack("H4")[0]

# unsupported format

if header != "21a8" then

unsupported\_format(raw)

return

end

# supported format

seq = raw[2].unpack("H2")[0]

# PANID

myPanid = raw[3..4].unpack("S\*")[0]

# RX Address

rxAddr = raw[5..6].unpack("S\*")[0]

# TX Address

txAddr = raw[7..8].unpack("S\*")[0]

#

str\_buf = raw[9..len-2].unpack("Z\*")[0]

print(sprintf("PANID=0x%04X, rxAddr=0x%04X, txAddr=0x%04X, Strings:: %s",myPanid,rxAddr,txAddr,str\_buf))

end

# printing unsupported format

def unsupported\_format(raw)

data = raw.unpack("H\*")

print("unsupported format::",data,"\n")

end

end

Rawデータをモニターに出力

ドライバからrawデータを取得

デバイスドライバを開く

Ctrl+Cを押したときの処理

デバイスファイルの指定

##

# Main function

##

class MainFunction

### Variable definition

bp3596\_dev = "/dev/bp3596" # Receiving pipe

finish\_flag = 0 # Finish flag

# Process at the time of SIGINT Receiving

Signal.trap(:INT){

finish\_flag=1

}

# Open the Receiving pipe

bp3596\_fp = open(bp3596\_dev, "rb")

bp\_api = BP3596PipeApi.new

# Loop until it receives a SIGINT

while finish\_flag==0 do

# Read device

recv\_buf = bp\_api.read\_device(bp3596\_fp)

if recv\_buf == -1

next

end

# Create a transmit buffer from the receive buffer

bp\_api.print\_raw2(recv\_buf)

end

# Close the Receiving pipe

bp3596\_fp.close

end