µracoli Arduino-Support-Package

Version 0.5.20170508

Generated by Doxygen 1.8.11

Contents

1 Main Page

This pages describe how the Arduino Environment is used together with the libraries of the µracoli project.

µracoli provides a support package for the Arduino-IDE. The UASP (µracoli-arduino-support-package) includes a Arduino-Serial-like library for the radio transceiver and board definitions for some selected boards.

2 Boards and Modules

2.1 atzb256rfr2xpro

Board #1: Atmel ATmega256RFR2 ZigBit Xplained Pro Extension

Parameters

- Build Target: `atzb256rfr2xpro`
- Build Aliases: secatrcb256rfr2xpro
- Include file: `board_derfa.h`
- Baudrate: `57600`
- MMCU: `atmega256rfr2`
- F_CPU: `16000000UL`
- Provides: `hif, key, led, mcu_t, mcu_vtg, rtc, sensors, tmr, trx`

Applications

rdiag, rsensor, selftest, sniffer, wgpio, wibo, wibohost, wibotest, wuart

Examples

xmpl_dbg, xmpl_hif, xmpl_key_events, xmpl_keys, xmpl_leds, xmpl_linbuf_rx, xmpl_linbuf_tx, xmpl_radio_↩ range, xmpl_radio_stream, xmpl_rtc, xmpl_sensor, xmpl_timer, xmpl_timer_callback, xmpl_trx_base, xmpl_trx_txrx, xmpl_trx_txrx_auto

2.2 mnb900

Board #2: Meshnetics MeshBean WDB-A1281 and MNZB-900 development boards

Parameters

- Build Target: `mnb900`
- Include file: `board_wdba1281.h`
- Baudrate: `38400`
- MMCU: `atmega1281`
- F_CPU: `8000000UL`
- Provides: `ds18b20, hif, i2c, led, lm73, mcu_vtg, ow, rtc, sensors, tmr, trx, trxvtg, tsl2550`

Applications

rdiag, rsensor, selftest, sniffer, wgpio, wibo, wibohost, wibotest, wuart

Examples

xmpl_dbg, xmpl_hif, xmpl_i2c, xmpl_leds, xmpl_linbuf_rx, xmpl_linbuf_tx, xmpl_lm73, xmpl_ow, xmpl_radio_↩ range, xmpl_radio_stream, xmpl_rtc, xmpl_sensor, xmpl_timer, xmpl_timer_callback, xmpl_trx_base, xmpl_trx_txrx, xmpl_trx_txrx_auto, xmpl_tsl2550

2.3 pinoccio

Board #3: Pinoccio - the ecosystem for the internet of things

Parameters

- Build Target: `pinoccio`
- Include file: `board_derfa.h`
- Baudrate: `115200`
- MMCU: `atmega256rfr2`
- F_CPU: `16000000UL`
- Fuses: `-U lfuse:w:0xf7:m -U hfuse:w:0xda:m -U efuse:w:0xf5:m`

- Provides: `hif, led, mcu_t, mcu_vtg, rtc, sensors, tmr, trx`

Applications

rdiag, rsensor, selftest, sniffer, wgpio, wibo, wibohost, wibotest, wuart

Examples

xmpl_dbg, xmpl_hif, xmpl_leds, xmpl_linbuf_rx, xmpl_linbuf_tx, xmpl_radio_range, xmpl_radio_stream, xmpl_rtc, xmpl_sensor, xmpl_timer, xmpl_timer_callback, xmpl_trx_base, xmpl_trx_txrx, xmpl_trx_txrx_auto

2.4 radiofaro

Board #4: RadioFaro, Arduino like board with deRFmega128-22A001

Parameters

- Build Target: `radiofaro`
- Include file: `board_radiofaro.h`
- Baudrate: `57600`
- MMCU: `atmega128rfa1`
- F_CPU: `16000000UL`
- Fuses: `-U lfuse:w:0xf7:m -U hfuse:w:0x9a:m -U efuse:w:0xfe:m`
- Provides: `hif, led, mcu_t, mcu_vtg, rtc, sensors, tmr, trx, trxvtg`

Applications

rdiag, rsensor, selftest, sniffer, wgpio, wibo, wibohost, wibotest, wuart

Examples

xmpl_dbg, xmpl_hif, xmpl_leds, xmpl_linbuf_rx, xmpl_linbuf_tx, xmpl_radio_range, xmpl_radio_stream, xmpl_rtc, xmpl_sensor, xmpl_timer, xmpl_timer_callback, xmpl_trx_base, xmpl_trx_txrx, xmpl_trx_txrx_auto

2.5 wdba1281

Board #5: Meshnetics MeshBean WDB-A1281 and MNZB-900 development boards

Parameters

- Build Target: `wdba1281`
- Build Aliases: mnb900
- Include file: `board_wdba1281.h`
- Baudrate: `38400`
- MMCU: `atmega1281`
- F_CPU: `8000000UL`
- Provides: `ds18b20, hif, i2c, led, lm73, mcu_vtg, ow, rtc, sensors, tmr, trx, trxvtg, tsl2550`

Applications

rdiag, rsensor, selftest, sniffer, wgpio, wibo, wibohost, wibotest, wuart

Examples

xmpl_dbg, xmpl_hif, xmpl_i2c, xmpl_leds, xmpl_linbuf_rx, xmpl_linbuf_tx, xmpl_lm73, xmpl_ow, xmpl_radio_↩range, xmpl_radio_stream, xmpl_rtc, xmpl_sensor, xmpl_timer, xmpl_timer_callback, xmpl_trx_base, xmpl_trx_txrx, xmpl_trx_txrx_auto, xmpl_tsl2550

3 Arduino IDE support

3.1 Overview

`Arduino` is a Microcontroller Development plattform that consists of a Java-IDE which supports various microcontroller boards. Most of the supported boards are equipped mit 8-bit-AVR microcontrollers.

Basically the Arduino-IDE provides a simple code editor and the firmware (denoted as "sketches") can be compiled and flashed with a button click. A serial terminal completes the IDE. This high level of abstration makes it very easy for none technicians and first-time users to start with embedded programming.

A **sketch** basically implements two functions `setup()` and `loop()` that are called from the main function of the core library. A simple API is provided, that is described at `http://arduino.cc/en/Reference/Home↩Page`.

µracoli provides a support package package for the Arduino-IDE with versions above 1.5.x.

The following boards are supported in the package:

- radiofaro - Radiofaro w/ ATmega128rfa1
- pinoccio - Pinoccio Scout
- atzb256rfr2xpro - Atmel ATmega256RFR2 Xplained Pro Evaluation Kit
- wdba1281 - Zigbit 2400MHz, w/ ATmega1281
- mnb900 - Zigbit 900MHz, w/ ATmega1281

The Arduino project did fork in 2014, see http://hackaday.com/2015/04/06/arduino-ide-forked/ for details. Since this time there are two versions of the IDE avaiable.

- IDE from arduino.cc
  https://www.arduino.cc/en/Main/Software
- IDE from arduio.org
  http://www.arduino.org/downloads

Note

In version arduino.cc-v1.6.5/arduino.org-v1.7.3 the arduino.org IDE comes with a complete arm-none-eabi GNU toolchain and a CMSIS for Atmel Cortex-M0/M0+/M4 controllers. The arduino.cc IDE is the SAM and SAMD board variants are missing too. The rest of both IDE-packages seems to be identical.

3.2 Installation

Download and install one of the Arduino IDEs.

Select one of the forked IDEs

- IDE from arduino.cc
  https://www.arduino.cc/en/Main/Software
- IDE from arduio.org
  http://www.arduino.org/downloads

and install it on your computer according to the instructions for your OS (Linux, Windows, MAC-OS).

Download the µracoli-Arduino-Support-Package (UASP)

Download the UASP-zipfile uracoli-arduino-15x- <version>.zip from http://uracoli.nongnu.org/download.html

Option A: Install UASP in the IDE directory

With this method the package is installed centrally for all users. The UASP example sketches are located in "File / Examples / Radio"

- Change to the Arduino-IDE directory (e.g. "cd /opt/arduino-1.6.5/")
- Unpack the UASP (e.g. "unzip uracoli-arduino-15x-<version>.zip")
  The UASP has three top level directories, hardware, examples, doc) the fell directly in the existing Arduino-IDE directories.
- Open the Arduino-IDE (e.g. "./arduino").
- If you see in the menu "Tools / Board" the boards Radiofaro, Zigbit2400 and Zigbit900, the installation was successful.

Option B: Install UASP in the Sketchbook folder

This method installs the UASP locally in the sketchbook folder of the current user. The UASP example sketches are located in "File / Sketchbook / Radio"

- Open the Arduino IDE (e.g. "/opt/arduino-1.6.5/arduino")
- Open the menu item "Files / Preferences")
  a dialogue window pops up. Here you find the location of the "Sketchbook location" in the first entry.
- Change to the "Sketchbook location", e.g. "cd /home/axel/Arduino".
- Unpack the UASP (e.g. "unzip uracoli-arduino-15x-<version>.zip")
  The UASP has three top level directories, hardware, examples, doc) the fell directly in the existing Arduino-IDE directories.
- Reopen the Arduino-IDE (e.g. "/opt/arduino-1.6.5/arduino") to load the UASP boards and examples into the IDE.
- If you see in the menu "Tools / Board" the boards Radiofaro, Zigbit2400 and Zigbit900, the installation was successful.

3.3 Usage

Using the HelloRadio sketch

Restart the Arduino-IDE after unpacking uracoli-arduino- <version>.zip and check if you see the new boards at the end of the list that opens if you click the menu entry "Tools/Board" and select your radio board, e.g. "Radiofaro", "Zigbit 2400MHz", etc.

In the next step select the in the menu "Tools / Serial Port" the serial port to which your radio board is connected.

Now open the HelloRadio sketch. It can be found either in "File / Examples / Radio" or "File / Sketchbook / Radio", depending on the installation option you did choose (see Installation).

Now select the menu entry "File / Upload". The sketch will compiled and flashed to the choosen radio board.

Note

If the upload fails, mark the options "`compile verbose`" and "`upload verbose`" in the dialogue that opens when clicking "`File / Preferences`" and check in the lower window pane of the Arduino-IDE what goes wrong.

The sketch "HelloRadio", that you have currently uploaded, sends a short frame every 500ms on channel 17 and reports the transmission also on its serial port. You can open a terminal with "`Tools / SerialMonitor`" and you should see in the terminal window:

```
HelloRadio
TX: 0
TX: 1
TX: 2
...
```

Each printed number shows, that a frame was successfully transmitted. If you see this output, that means that you are now "on air".

Example Sketches

A good starting point for using the radio functions are the example sketches.

- IoCheck.ino
- RadioUart.ino
- HelloRadio.ino
- IoRadio.ino
- Gateway.ino
- Remote.ino

Function Reference

The regular Arduino core functions are documented at `http://arduino.cc/en/Reference/HomePage`.

The radio specific functions are described in section Arduino Radio Functions.

Building Sketches from Command Line

Beside several thirdparty CLI tools, e.g. Arscons, Inotool, since version 1.5 the Arduino IDE supports its own CLI, see `https://github.com/arduino/Arduino/blob/ide-1.5.x/build/shared/manpage.adoc`

```
arduino --verify \
        --board uracoli:avr:radiofaro \
        examples/Radio/RadioUart/RadioUart.ino
```

Note

- In order to explicitely set the Arduino build path add `-pref` build.path=<your_build_dir> to the command.
- With option -v, a more verbose output of the build process is generated and the current Arduino build directory is shown.

3.4 Bootloader

If the Arduino Bootloader in your board is accidently erased, you can restore it with the following procedure.

In order to flash the bootloader, use a flash programmer and a programming tool of your choice (avrdude, atprogram, Atmel-Studio) and flash the file to the board.

The source code and a precompiled Intel-Hex file of the bootloader is located in the directory `hardware/uracoli/avr/bootloa`

Check also if the fuses are set correctly:

```
LF = 0xe7
HF = 0x90
EF = 0xf6
```

Examples

The preferred flashing program is avrdude since it comes with the Arduino distribution.

Note

: If you use avrdude from the arduino package, an error message about the missing config file may occur.

```
avrdude: can't open config file "...avrdude.conf": No such file or directory
avrdude: error reading system wide configuration file
```

So you have to give the path to the config file explicitely.

Examples

```
# Connection AVR dragon via isp
PROG=avrdragon_isp
# Connection AVR dragon via jtag
PROG=avrdragon_jtag

avrdude -C $ARDUIONDIR/hardware/tools/avr/etc/avrdude.conf \
        -P usb -c $PROG -p m128rfa1 \
        -U lf:w:0xe7:m -U hf:w:0x90:m -U ef:w:0xf7:m \
        -U fl:w:ATmegaBOOT_radiofaro.hex
```

If the bootloader is flashed correctly, it can be checked with

```
avrdude   -C $ARDUIONDIR/hardware/tools/avr/etc/avrdude.conf \
          -P <MYSERIALPORT> \
          -b 57600 -c arduino -p m128rfa1 -U <MYHEXFILE>
```

3.5 Licenses

This package incorporates source code from different license models, which has an influence on the usae of the code in proprieatary projects and environments.

GPL version 2.0

According to the file header, the bootloader is licensed under GNU General Public License version 2.0. See http://www.gnu.org/licenses/gpl-2.0.txt or file link:license_gpl_2v0.txt[].

```
hardware/uracoli/bootloaders/radiofaro/ATmegaBOOT.c
```

LGPL version 2.1

The files copied from the original Arduino core are licensed under the GNU Lesser General Public License version 2.1. This code is linked to each sketch. See http://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt or file link:license_lgpl_2v1.txt[].

```
hardware/uracoli/variants/zigbit900/pins_arduino.h
hardware/uracoli/variants/zigbit2400/pins_arduino.h
hardware/uracoli/variants/radiofaro/pins_arduino.h
```

Modified BSD license

The sources of the Âµracoli radio functions are licensed under the modified 3 clause BSD license. See file link:license_uracoli.txt[].

```
hardware/uracoli/cores/uracoli/trx_rf230_param.c
hardware/uracoli/cores/uracoli/const.h
hardware/uracoli/cores/uracoli/boards/base_zdma1281.h
hardware/uracoli/cores/uracoli/boards/board_derfa.h
hardware/uracoli/cores/uracoli/boards/board_wdba1281.h
hardware/uracoli/cores/uracoli/trx_rf230_sram.c
hardware/uracoli/cores/uracoli/trx_datarate_str.c
hardware/uracoli/cores/uracoli/board.h
hardware/uracoli/cores/uracoli/at86rf230b.h
hardware/uracoli/cores/uracoli/trx_rf230.c
hardware/uracoli/cores/uracoli/at86rf212.h
hardware/uracoli/cores/uracoli/atmega_rfa1.h
hardware/uracoli/cores/uracoli/trx_datarate.c
hardware/uracoli/cores/uracoli/trx_rf230_frame.c
hardware/uracoli/cores/uracoli/trx_rfa.c
hardware/uracoli/cores/uracoli/trx_rf230_irq.c
hardware/uracoli/cores/uracoli/trx_rf230_bitwr.c
hardware/uracoli/cores/uracoli/trx_rf230_bitrd.c
hardware/uracoli/cores/uracoli/radio_rf230.c
hardware/uracoli/cores/uracoli/at86rf230a.h
hardware/uracoli/cores/uracoli/radio_rfa.c
hardware/uracoli/cores/uracoli/radio.h
hardware/uracoli/cores/uracoli/trx_rf230_crc.c
hardware/uracoli/cores/uracoli/transceiver.h
hardware/uracoli/cores/uracoli/usr_radio_irq.c
hardware/uracoli/cores/uracoli/trx_rf230_misc.c
examples/Radio/RadioUart/RadioUart.ino
examples/Radio/IoCheck/IoCheck.ino
examples/Radio/HelloRadio/HelloRadio.ino
examples/Radio/IoRadio/IoRadio.ino
```

4 Module Documentation

4.1 Arduino Radio Functions

Description of the UASP functions.

Data Structures

- struct radio_buffer_t
- class HardwareRadio

4.1.1 Detailed Description

4.1.2 Data Structure Documentation

4.1.2.1 struct radio_buffer_t

Element of a chained list of frame buffers

Data Fields

uint8_t	frm[PHY_MAX_FRAME_SIZE]	array that can store a maximum IEEE 802.15.4 frame
uint8_t	idx	read/write index
uint8_t	len	Length of payload
struct radio_buffer *	next	pointer to next list element or NULL if list terminates

4.1.2.2 class HardwareRadio

Hardware Radio class

Public Member Functions

- radio_buffer_t * alloc_buffer (void)
- void free_buffer (radio_buffer_t *pbuf)
- HardwareRadio (void)
- void begin (void)
- void begin (uint8_t channel, uint8_t idlestate)
- void begin (uint8_t channel, uint8_t idlestate, uint16_t pan, uint16_t dst, uint16_t src)
- virtual int available (void)
- virtual int peek (void)
- virtual int read (void)
- virtual void flush (void)
- virtual size_t write (uint8_t)
- void write (char *str)
- void write (uint8_t *buf, uint8_t size)
- uint8_t sendto (uint16_t dst, uint8_t *pbuf, uint8_t size)

Constructor & Destructor Documentation

4.1.2.2.1 HardwareRadio::HardwareRadio (void)

constructor

Member Function Documentation

4.1.2.2.2 radio_buffer_t* HardwareRadio::alloc_buffer (void)

Allocate a radio buffer

4.1.2.2.3 virtual int HardwareRadio::available (void) [virtual]

return number of available bytes in current RX buffer

Examples:

Gateway.ino, and RadioUart.ino.

4.1.2.2.4 void HardwareRadio::begin (void)

Starting the hardware radio class with default parameters

Note

The following default parameters are used implicitely.

- channel: PHY_DEFAULT_CHANNEL
- idlestate: STATE_RXAUTO
- pan id: DEFAULT_PAN_ID
- destination address: DEFAULT_SHORT_ADDRESS
- source address: DEFAULT_SHORT_ADDRESS

Examples:

Gateway.ino, HelloRadio.ino, IoCheck.ino, IoRadio.ino, RadioUart.ino, and Remote.ino.

4.1.2.2.5 void HardwareRadio::begin (uint8_t *channel*, uint8_t *idlestate*)

Starting the hardware radio class with explicit parameters

Parameters

channel	radio channel (11 - 26 for 2.4GHz radios, 0 - 10 for SubGHz radios)
idlestate	default state of the radio, supported values are listed in radio_state_t.

Note

The following default parameters are used implicitely.

- pan id: DEFAULT_PAN_ID
- destination address: DEFAULT_SHORT_ADDRESS
- source address: DEFAULT_SHORT_ADDRESS

4.1.2.2.6 void HardwareRadio::begin (uint8_t *channel,* uint8_t *idlestate,* uint16_t *pan,* uint16_t *dst,* uint16_t *src*)

Starting the hardware radio class with explicit parameters

Parameters

channel	radio channel (11 - 26 for 2.4GHz radios, 0 - 10 for SubGHz radios)
idlestate	default state of the radio, supported values are listed in radio_state_t.
pan	_p_ersonal _a_rea _n_etwor ID
dst	16 bit destination address
src	16 bit node address

4.1.2.2.7 virtual void HardwareRadio::flush (void) [virtual]

flush TX and RX queues. RX queue data are discarded, TX data is sent.

Examples:

HelloRadio.ino, IoRadio.ino, RadioUart.ino, and Remote.ino.

4.1.2.2.8 void HardwareRadio::free_buffer (radio_buffer_t * *pbuf*)

Free a radio buffer

4.1.2.2.9 virtual int HardwareRadio::peek (void) [virtual]

Returns the next byte (character) of incoming data (RX) without removing it from the internal serial buffer.

Returns

EOF (-1) if no data available, otherwise a value from 0 ... 255

4.1.2.2.10 virtual int HardwareRadio::read (void) [virtual]

Returns the next byte (character) of incoming data (RX)

Returns

EOF (-1) if no data available, otherwise a value from 0 ... 255

Examples:

Gateway.ino, and RadioUart.ino.

4.1.2.2.11 uint8_t HardwareRadio::sendto (uint16_t *dst,* uint8_t * *pbuf,* uint8_t *size*)

send binary data direct to a node, adressed by dst.

Parameters

dst	destination address
pbuf	pointer to data array
size	number of bytes to transmit

Returns

number of bytes transmitted

Note

At the end of this routine flush is called, and therefore the data collected in write and and print are sent out to.

4.1.2.2.12 virtual size_t HardwareRadio::write (uint8_t) [virtual]

write a byte to the TX stream

Examples:

HelloRadio.ino, and IoRadio.ino.

4.1.2.2.13 void HardwareRadio::write (char * *str*)

write a string to the TX stream

Parameters

str	\0 terminated string

4.1.2.2.14 void HardwareRadio::write (uint8_t * *buf,* uint8_t *size*)

write a binary buffer (buf, size) to the TX stream

Parameters

buf	pointer to the buffer
size	number of bytes in the buffer.

4.1.3 Defines

4.1.3.1 #define PHY_DEFAULT_CHANNEL (17)

Default radio channel number in 2.4 GHz band

4.1.3.2 #define PHY_MAX_CHANNEL (26)

Maximun radio channel number in 2.4 GHz band

4.1.3.3 #define PHY_MAX_FRAME_SIZE (127)

Maximum size of a IEEE 802.15.4 frame

4.1.3.4 #define PHY_MIN_CHANNEL (11)

Minimun radio channel number in 2.4 GHz band

5 Example Documentation

5.1 Gateway.ino

This sketch implements a COSM Gateway.

```
/*
  Cosm temperature logger

  This sketch is derived from PachubeClientString Example bundled with
  Arduino. It receives a character line from temperature sensor and feeds
  it to cosm.

  Hardware:
   - Radiofaro + DFRobot Ethernet Shield attached

 */

#include <SPI.h>
#include <Ethernet.h>
#include "HardwareRadio.h"


#define APIKEY         "The quick brown fox" // replace your Pachube api key here
#define FEEDID         123456 // replace your feed ID
#define USERAGENT      "Radiofaro" // user agent is the project name


// assign a MAC address for the ethernet controller.
// fill in your address here:
  byte mac[] = {  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

// fill in an available IP address on your network here,
// for manual configuration:
IPAddress ip(10,0,1,20);

// initialize the library instance:
EthernetClient client;

// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
IPAddress server(216,52,233,121);      // numeric IP for api.pachube.com
//char server[] = "api.pachube.com";   // name address for pachube API

unsigned long lastConnectionTime = 0;          // last time you connected to the server, in milliseconds
boolean lastConnected = false;                 // state of the connection last time through the main loop
const unsigned long postingInterval = 20*1000;  //delay between updates to pachube.com

char ln[80]; /* max line length */
byte idx = 0;
byte lncnt = 0;
boolean hasData=0;

String dataString = "";

String addr_whitelist[] = {"0xFECA", "0xAFFE"};

void setup() {
```

```
  Radio.begin();
  Serial.begin(9600);

  // give the ethernet module time to boot up:
  delay(1000);
  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // DHCP failed, so use a fixed IP address:
    Ethernet.begin(mac, ip);
  }
}

boolean parse(char *ln)
{
    byte i=0;
    char *sarr[8];
    char *s = ln;
    boolean valid=0;

    Serial.print("Line In: ");
    Serial.print(ln); // line break already in line

    /* tokenize to all space separated */
    do{
      if((*s==',') || (*s == '=')) *s=' ';
    }while(*s++);

    s=sarr[i++]=strtok(ln, " ");
    dataString="";
    do{
      s=sarr[i++]=strtok(NULL, " ");
    }while( (NULL != s) && (i<sizeof(sarr)/sizeof(sarr[0])) );

    if(sarr[2][0]=='T' && sarr[4][0]=='V'){

        s=sarr[3];
        while(*s && (*s>='0' && *s<='9' || *s=='.' || *s=='-') ){
          s++;
        };

        if(*s){ /* did not reach NULL terminator */
          /* N.a.N. */
        } else {
          dataString += "temp,";
          dataString += sarr[3];
          dataString += "\nvolt,";
          dataString += sarr[5];
          valid = 1;
        }
    }

    if(!valid){
       Serial.println("Parsing error");
    } else {
      Serial.print("Parsed: ");
      Serial.println(dataString);
    }
    return valid;
}

void loop() {

  if (Radio.available() > 0)
  {
    char c = Radio.read();

    if(c == '\0'){ /* unexpected input, restart */
      idx = 0;
    } else {
      ln[idx] = c;
      idx++;
    }
    ln[idx] = '\0';

    if( ('\n' == c) || ('\r' == c) ) { /* line completed */
      hasData = parse(ln);

      lncnt++;
      idx = 0;
    }
  }

  // if there's incoming data from the net connection.
  // send it out the serial port.  This is for debugging
  // purposes only:
  if (client.available()) {
```

```
    char c = client.read();
    Serial.print(c);
  }

  // if there's no net connection, but there was one last time
  // through the loop, then stop the client:
  if (!client.connected() && lastConnected) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }

  // if you're not connected, and ten seconds have passed since
  // your last connection, then connect again and send data:
  if(!client.connected() && (millis() - lastConnectionTime > postingInterval) && hasData>0) {
    Serial.print("Pachube: ");
    Serial.println(dataString);

    sendData(dataString);
    hasData=0;
  }
  // store the state of the connection for next time through
  // the loop:
  lastConnected = client.connected();
}

// this method makes a HTTP connection to the server:
void sendData(String thisData) {
  // if there's a successful connection:
  if (client.connect(server, 80)) {
    Serial.println("connecting...");
    // send the HTTP PUT request:
    client.print("PUT /v2/feeds/");
    client.print(FEEDID);
    client.println(".csv HTTP/1.1");
    client.println("Host: api.pachube.com");
    client.print("X-pachubeApiKey: ");
    client.println(APIKEY);
    client.print("User-Agent: ");
    client.println(USERAGENT);
    client.print("Content-Length: ");
    client.println(thisData.length());

    // last pieces of the HTTP PUT request:
    client.println("Content-Type: text/csv");
    client.println("Connection: close");
    client.println();

    // here's the actual content of the PUT request:
    client.println(thisData);
  }
  else {
    // if you couldn't make a connection:
    Serial.println("connection failed");
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }
  // note the time that the connection was made or attempted:
  lastConnectionTime = millis();
}
```

5.2 HelloRadio.ino

This sketch sends frames with a period of 500ms.

```
/* $Id$ */
#include "HardwareRadio.h"

#define REG(name, reg) do{uint8_t _b_ = reg; Serial.print(name);Serial.print(" : ");
       Serial.println(_b_,HEX);}while(0)

unsigned long tx_time;
int cnt = 0;
void setup() {

    /* operating on channel 17, not receiving when idle. */
    Radio.begin(17, STATE_OFF);
    Serial.begin(57600);
    Serial.println("HelloRadio V$Release$");
```

```
}

void loop() {

    if (millis() > tx_time){
        tx_time = millis() + 500;
        // write string
        Radio.write("cnt: ");
        // write integer
        Radio.print(cnt);
        // write byte
        Radio.write('\n');
        Radio.flush();
        Serial.println(cnt);
        cnt ++;
    }

}
```

5.3 IoCheck.ino

This is sketch performs a simple GPIO check, no radio functions included.

```
/* $Id$ */
#include "HardwareRadio.h"

bool DO_ECHO = false;

void setup()
{

    Serial.begin(57600);
    // flush input
    while(Serial.available())
    {
        Serial.read();
    }
    Radio.begin();
    Serial.println("Sketch IoCheck\n\r"
                   "Type help for a list of commands");
}

void loop()
{
    static uint8_t val = 0;
    static char line[32], *pcmd;

    if (DO_ECHO)
    {
        Serial.print("\n\rIoCheck>");
    }
    /* check serial IO */
    pcmd = readline(line, sizeof(line));

    if(pcmd != NULL)
    {
        if (DO_ECHO)
        {
            Serial.println(pcmd);
        }
        process_command(pcmd);
    }
}

// ========================================================
// parse command and dispatch to execution function
void process_command(char *pcmd)
{
    char *argv[8], *p;
    int argc = 0, i;
    bool newarg;

    p = pcmd;
    argv[argc++] = p;
    newarg = false;

    while(*p != 0)
    {
        if (*p == ' ')
```

```
        {
            *p = 0;
            newarg = true;
        }
        else
        {
            if (newarg == true && argc < 8)
            {
                argv[argc++] = p;
                newarg = false;
            }
        }
        p++;
    }
    #if 0
    // dump result of line parsing
    for (i=0; i<argc; i++)
    {
        Serial.print(" - ");
        Serial.print(i);
        Serial.print(":");
        Serial.println(argv[i]);
    }
    #endif

    if (strncasecmp("pin", argv[0], 3) == 0)
    {
        execute_pin_command(argv + 1, argc - 1);
    }
    else if (strncasecmp("help", argv[0], 4) == 0)
    {
        execute_print_help();
    }
    else if (strncasecmp("echo", argv[0], 4) == 0)
    {
        DO_ECHO = atoi(argv[1]) ? true : false;
        Serial.print("echo is now ");
        Serial.println(DO_ECHO ? "ON": "OFF");
    }
    else if (strncasecmp("jtag", argv[0], 4) == 0)
    {
        execute_set_jtag(atoi(argv[1]) ? true : false);
    }
    else if (strncasecmp("sleep", argv[0], 4) == 0)
    {
        execute_set_sleep();
    }
    else if (strncasecmp("tx", argv[0], 2) == 0)
    {
        execute_transmit();
    }
    else if (strncasecmp("rx", argv[0], 2) == 0)
    {
        execute_receive(atoi(argv[1]));
    }
}

void execute_print_help(void)
{
    Serial.print("\n\r"
        " pin <id> <dir> <val> : set pin (id is e.g. 'd0' or 'a0', dir: 0=input, 1=output)\n\r"
        " echo <bool>          : switch echo ON/OFF\n\r"
        " jtag <bool>          : switch jtag ON/OFF\n\r"
        " sleep                : set MCU and TRX to sleep\n\r"
        " tx                   : transmit frame\n\r"
        " rx <tmo>             : receive frame (tmo to wait for frame in ms)\n\r"
        " help                 : print help\n\r");
}
```

5.4 IoRadio.ino

This sketch reads A0 and A1 each 500ms, the results are printed to serial terminal and to the Radio.

```
/* $Id$ */
#include <stdio.h>
#include "HardwareRadio.h"

#define REG(name, reg) do{uint8_t _b_ = reg; Serial.print(name);Serial.print(" : ");
       Serial.println(_b_,HEX);}while(0)

unsigned long tx_time;
```

```
int cnt = 0;
void setup() {

    /* operating on channel 17, not receiving when idle. */
    Radio.begin(17, STATE_OFF);
    Serial.begin(57600);
    Serial.println("RadioIo");
}

void loop() {
int a0, a1;
char buf[64];

    if (millis() > tx_time)
    {
       a0 = analogRead(0);
       a1 = analogRead(1);
       snprintf(buf, 64, " a0: %d, a1: %d\r\n", a0, a1);

       // sent wireless
       Radio.write("cnt: ");
       Radio.print(cnt);
       Radio.write(buf);
       Radio.flush();

       // local echo of results
       Serial.print("cnt:");
       Serial.print(cnt);
       Serial.print(buf);

       tx_time = millis() + 500;
       cnt ++;
    }

}
```

5.5 RadioUart.ino

This sketch implements a wireless UART application.

```
/* $Id$ */
/* RadioFaro wirless/serial bridge. */
#include "HardwareRadio.h"

/* data captured from UART */
char serial_inbyte = 0;

/* data received from Radio */
char serial_outbyte = 0;

/* next time the radio buffer will flushed */
unsigned long tx_time;

void setup() {
    Radio.begin();
    Serial.begin(57600);
    tx_time = 0;
    Serial.println("RadioUart.ino");
}

void loop() {

    if ((Serial.available() > 0))
    {
        serial_inbyte = Serial.read();
        Radio.print(serial_inbyte);
    }

    if (millis() > tx_time){
        tx_time = millis() + 25;
        Radio.flush();
    }

    if (Radio.available() > 0)
    {
        serial_outbyte = Radio.read();
        Serial.print(serial_outbyte);
    }
}
```

5.6 Remote.ino

This sketch implements a COSM remote sensor, that measures temperature and VCC. The board sleeps during the idle period and is woken up by the watch dog.

```
/*
  Cosm temperature logger

  Remote node application

  Hardware:
   - Radiofaro on battery power, temperature sensor internal
*/

#include <avr/wdt.h>
#include <avr/sleep.h>
#include "HardwareRadio.h"

/* === globals ============================================================= */
static volatile uint8_t adcfinished = 0;
static int8_t adc_offset = 0;

/* === functions =========================================================== */

/*
 * \brief Setup watchdog to serve as application timer
 *
 */
static inline void wdt_timersetup(uint8_t timeout)
{
    WDTCSR = (1 << WDCE) | (1 << WDE); /* Enable configuration change */
    WDTCSR = (1 << WDIF) | (1 << WDIE) | /* Enable Watchdog Interrupt Mode */
    (timeout & 0x08 ? _WD_PS3_MASK : 0x00) | (timeout & 0x07);
}

/*
 * \brief Watchdog ISR, used as application timer
 *
 */
ISR(WDT_vect, ISR_NOBLOCK)
{
  /* do nothing, just wake up MCU */
}

ISR(ADC_vect, ISR_BLOCK)
{
    adcfinished = 1;
}

/*
 * \brief Trigger sleep based ADC measurement
 * Function is blocking until flag "adcfinished" is set by ISR
 *
 * @return ADC register value
 */
static inline int16_t adc_measure(void)
{
    set_sleep_mode(SLEEP_MODE_ADC);
    /* dummy cycle */
    adcfinished = 0;
    do
    {
        sleep_mode();
        /* sleep, wake up by ADC IRQ */

        /* check here for ADC IRQ because sleep mode could wake up from
         * another source too
         */
    } while (0 == adcfinished); /* set by ISR */
    return ADC ;
}

float measure_tmcu(void)
{
    int32_t val = 0;
    uint8_t nbavg = 5;

    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1); /* PS 64 */
    ADCSRB = (1 << MUX5);

    ADMUX = (1 << REFS1) | (1 << REFS0) | (1 << MUX3) | (1 << MUX0); /* reference: 1.6V, input Temp Sensor
      */
    _delay_us(200); /* some time to settle */
```

```
    ADCSRA |= (1 << ADIF); /* clear flag */
    ADCSRA |= (1 << ADIE);

    /* dummy cycle after REF change (suggested by datasheet) */
    adc_measure();

    _delay_us(100); /* some time to settle */

    for(uint8_t i=0;i<nbavg;i++){
        val += adc_measure() - adc_offset;
    }

    ADCSRA &= ~((1 << ADEN) | (1 << ADIE));

    return (1.13 * val / (float)nbavg - 272.8);
}

int8_t measure_adc_offset(void)
{
    uint16_t val;

    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1); /* PS 64 */
    ADCSRB = 0;
    ADMUX = (1 << REFS1) | (1 << REFS0) | (1 << MUX3); /* reference: 1.6V, differential ADC0-ADC0 10x */

    _delay_us(200); /* some time to settle */

    ADCSRA |= (1 << ADIF); /* clear flag */
    ADCSRA |= (1 << ADIE);

    /* dummy cycle after REF change (suggested by datasheet) */
    adc_measure();

    _delay_us(100); /* some time to settle */

    val = adc_measure();

    ADCSRA &= ~((1 << ADEN) | (1 << ADIE));

    return (val);
}

/*
 * \brief Measure internal voltage of ATmega128RFA1
 *
 * Cannot be measured via ADC, use Batmon of TRX part
 *
 */
float measure_vmcu(void)
{
    uint16_t v; /* voltage in [mV] */
    uint8_t vth;

    for(vth=0;vth<32;vth++){
        BATMON = vth & 0x1F;
        BATMON = vth & 0x1F;

        if(0 == (BATMON & (1<<BATMON_OK))){
            break;
        }
    }

    if(vth & 0x10){
        v = 2550 + 75*(vth&0x0F); /* high range */
    }else{
        v = 1700 + 50*(vth&0x0F); /* low range */
    }

    return v / 1000.0;
}

void setup()
{
  /* init all unused pins */
  DDRB = 0x00;  /* as input */
  PORTB = 0xFF; /* pullups */
  DDRD = 0x00;  /* as input */
  PORTD = 0xFF; /* pullups */
  DDRE = 0x00;  /* as input */
  PORTE = 0xFF; /* pullups */
  DDRF = 0x00;  /* as input */
  PORTF = 0xFF; /* pullups */
  DDRG = 0x00;  /* as input */
  PORTG = 0xFF; /* pullups */

  DIDR0 = 0xFF; /* disable all ADC inputs */
```

```
  /* we are using async Tx only, never receive
   * anything. Therefore we choose sleep as idle state
   */
  Radio.begin(17, STATE_SLEEP);

  adc_offset = measure_adc_offset();

  wdt_timersetup(WDTO_8S);
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
}

void loop()
{
  float tmcu, vmcu;

  tmcu = measure_tmcu();
  vmcu = measure_vmcu();

  Radio.print("ADDR=");
  Radio.print(Radio.nc.short_addr);

  Radio.print(", T=");
  Radio.print(tmcu);

  Radio.print(", V=");
  Radio.print(vmcu);

  Radio.print('\n');

  Radio.flush();
  while(Radio.tx_in_progress);

  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_mode();
}
```

Index