UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

"Development of wireless window sensor device integrated into a low power Home Automation System"

TRABAJO FINAL DE GRADO

Autor/a:

David Gómez Micó

Tutor/a:

Mª Asunción Pérez

GANDIA, 2014

Fachhochschule Düsseldorf

Development of wireless window sensor device integrated into a low power Home Automation System

Praxisprojekt

David Gómez Micó Matr.-Nr.664969

Prof. Dr.-Ing. Ulrich G. Schaarschmidt
B.Sc. Oliver von Fragstein

17/07/2014

Contents

Summary

This document is divided into two main blocks, Hardware and Software. The purpose is to explain in detail, all the process and the tools used for development the prototype named, wireless windows state sensor device.

In the first part, presents and explain the Atmega128RFA1 microcontroller and other hardware involved in the device.

In the second part, I talk about the software used in the process of Eagle for design the PCB and Atmel Studio and Contiki OS used in the program design

For concluding, I perform a functional test for the device verifying its correct performance.

Introduction

The purpose of this paper is to expose the process of Developing and production of a wireless windows state sensor device. This prototype is part of a low power home automation system which is based on Atmel ATmega128RFA1 microcontrollers using the OpenSource OS Contiki.

The device communicates with the central node via Wifi 802.11 and 6LoWPAN based on IEEE 802.15.4 with IPv6.

Software

Eagle

Eagle is used, because it’s a freeware software, have an easy environment of design and they have the option to easy design for libraries for new components needed in this project.

The version used in this document, is the EAGLE 5.11.00 (1). The libraries used in the design, it's a modification of the standard footprint for Atmega128RFA1 using a longest pads for easy weld in the PCB and the set Balun + Filter in the RF Antenna.

Atmel Studio

The manufacturer Atmel offers this integrated development platform (IDP) for develop software, its free and adds advanced features for debugging. In this project is used the version Atmel Studio 6.1 (2).

For use Contiki with Atmel Studio, need install Contiki and use an external tool for compile Contiki and create the files needs for programing the microcontroller. This part is explained in [Annex 1].

Contiki

Who is Contiki?

Contiki is a multitasking Operating System in real time. Contiki is characterized by using very little memory of the microprocessor stack, that others Operating System, for example the multithreading architecture. This OS has multiple parallel processes which are waiting to be activated conditions.

The processes when activated perform their tasks, when finish, stay waiting until other event call the process. The processes must be initialized so that they can be activated by events, and at any time can be destroyed. The events are the responsible for switch between different processes. These events are stored in a circular buffer and are processed in chronological order, from oldest to newest (3).

Moreover Contiki is an open source operating system, the development is easy and fast: Contiki applications are written in standard C.

Why Contiki?

It’s a wireless communication system using Open Source. The Wireless communication Works using 6LoWPAN protocol. This protocol is based on the creation of a small network that works over IP protocols. One of the main advantages of using protocols 6LoWPAN versus other wireless communication protocols, such as Zigbee, is the use of IP addresses directly.

The connection of the device to a Wi-Fi network it will be much easier and faster. Another important advantage is the free use of this OS, which is one of the main motivations choose Contiki for this project.

Processes

The Processes develop specific tasks and these are stored in a linked list, so only one need to know the address of the first. Each process has an associated protothread, which is executed when the process is called.

A protothread is a simple function that has special features. A process enters in called state, when stay running. Is on when it may be called (by events) and is deactivated when no event can wake the protothread.

Events

They are stored in a circular queue and are processed in chronological order. Each event has set the process that will awaken. These events can be generated in the protothreads or anywhere else.

Protothread:

They functions, are executed when the process that contains is called. These functions have the property that they are able to storing the return line through WaitForEvent. The next time an event call the process, its protothread will run from the stored line and ask if the event for which he woke up is one of the expected, if not it will return, otherwise the execution will continue the protothread until next WaitForEvent or until the end of protothread.

This gives the property protothread to wait for events.

Etimers

It's a module implemented by Contiki, for handling timers.

It's based on a linked list of structures called timers, these structures storing timespan. This timespan after which the timer will expire, have a pointer to the process that will awaken when the timer expires, and a pointer to eTimer for next on the list.

Moreover has an eTimer process, which is responsible for processing protothread list of timers. This list runs protothread timers and see if any expired, in which case the structure of the list is removed and an event "PROCESS_EVENT TIMER" is sent to the process that corresponds indicating that expired on eTimer.

Detailed Description

The event timers provide a way to generate timed events. When the timer expired, an event of time expired is generated, this call goes search the PROCESS WAIT_EVENT() associated (4).

An event timer is declared as a structure eTimer and all access to the event timer is made by a pointer to the declared event timer.

An example of all the concepts together (5)

```
PROCESS (blink_process, "blink_example");
AUTOSTART_PROCESSES(&Bblink_process);
PROCESS_THREAD(blink_process, ev data)
{
        PROCESS_BEGING();
        leds_off(LEDS_ALL);
        static struct etimer et;
        while(1){
                etimer_set(&et, CLOCK_SECKOND);

                PROCESS_WAIT_EVENT();
                leds_toggle(LEDS_GREEN);
        }
        PROCESS_END();
}
```

UML Diagram Windows Detector

The goal is a window opening detector using Contiki. For the design of the code will use the following algorithm, expressed in UML:

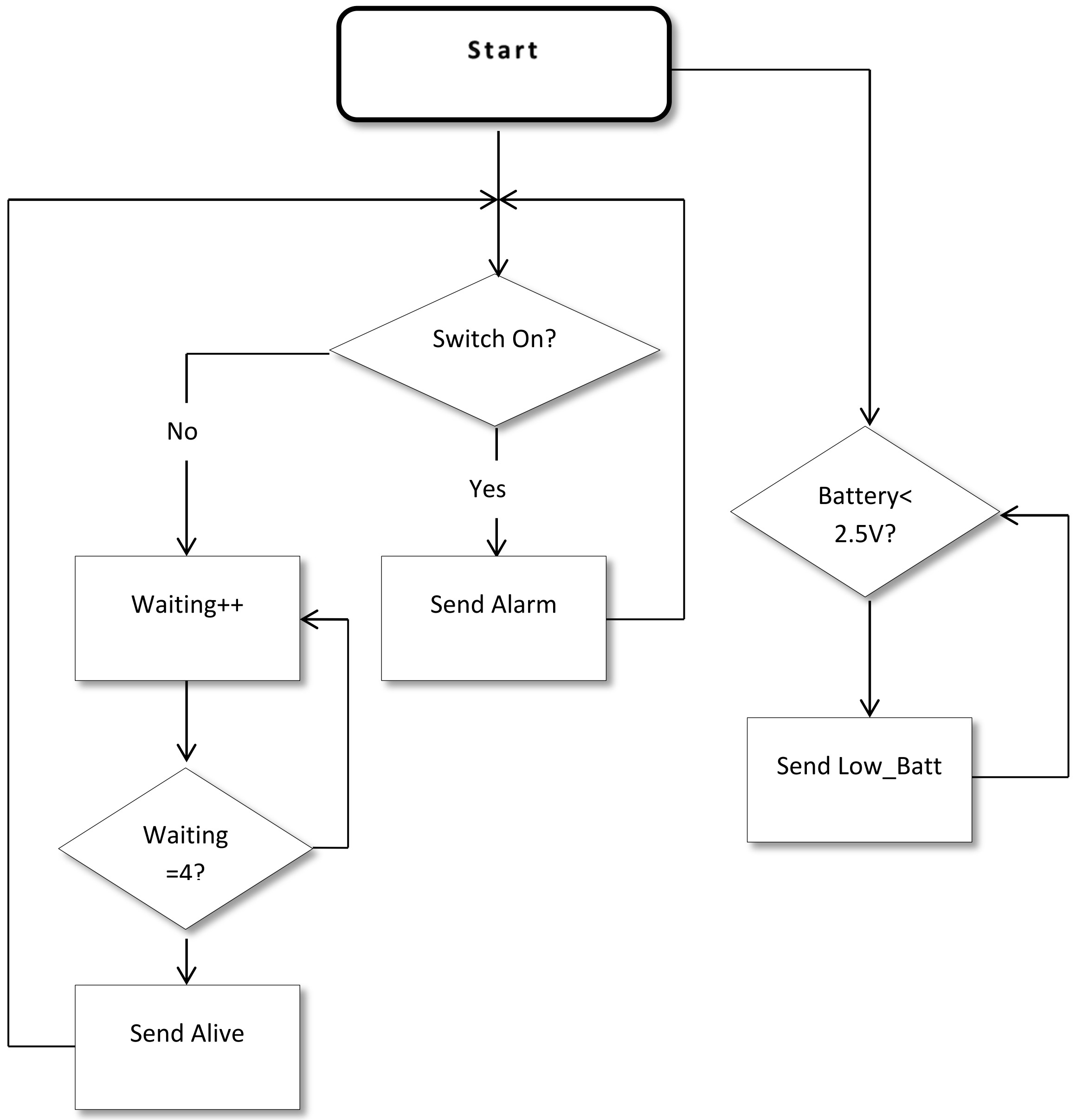


Figure 1: UML diagram

The Code

Using the UML, it's performed the code for implementing in the microcontroller with Contiki; the entire code can be found in [Annex 2]. Then, will be explained the most relevant parts of the code for an easy understanding.

The code is parameterized to modify timeouts for different eTimers

```
/* Definition Time Interval in Seconds*/
#define TIME_PERIODE     5          // Seconds for enter in Sleepmode
#define TIME_ALIVE       4          // Time for sending Alive
#define TIME_BATT        10         // Timer for sending Battery low
```

This part is for the configuration of the input and output ports as digital and configuration of the internal Pullup resistors.

```
        DDRF = 0xff; //Led
        DDRE = 0x00; //Sensor
        PORTE = 0xff; //Pull up On
```

This process, enable the connection to the central node using the simple_udp_register function. This function registers a UDP connection and attaches a callback function to it. The callback function will be called for incoming packets.

```
PROCESS_THREAD(Fensterdetektor, ev, data) {
/*. . . */
/* Regist UDP*/
  simple_udp_register(&multicast_connection, device.out,NULL, device.out,NULL);
  etimer_set(&timer, (CLOCK_SECOND * 3));
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

/* Multicast address as the recipient */
  uip_create_linklocal_allnodes_mcast(&addr);
```

The program checks the entry PE6 of the microcontroller, each TIME_PERIODE. If it's open sent to the central node, the information that the window been open Send_Alarm, if the window is closed, call the WAITING_STATE state, and the waiting_state variable increases, this variable controls the frequency of sent packet Alive.

The alarm flag and the alarm_timers are reset every time the windows are open.

```
/* Set the periodic timer */
  etimer_set(&timer_peri, CLOCK_SECOND * TIME_PERIODE);
while(1) {
            PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer_peri));
            if (PINE & (1<<PINE6)){ //If PF0 = 0 trigger condition, open window.
            PORTF &= ~(1 << 3);// Led On
            alarm_flag = ALARM;
            alive_timer = 0;
            waiting_state = 0;
            simple_udp_sendto(&multicast_connection, "p00d:fd:0010:alarm", 18, &addr);
            }else{
                        PORTF |= (1 << 3);
                        if(alarm_flag == ALARM){
                                    alarm_flag = WAITING_STATE;
                                                }
            }//else
            /*. . */
  }
  PROCESS_END();
}
```

These features make the control of the different states that the device sends. This part of the code it’s oriented to the energy efficiency of the device.

Batt_low: if the batteries are below 2.45 volts.
Alive: if the window is closed during "x" waiting_state.
Waiting: The window is closed.

```
/* Battery test? */
if(batt_timer >= TIME_BATT) {
            if(batt_check_threshold() == BELOW_THRESHOLD) {
            batt_timer = 0;
            simple_udp_sendto(&multicast_connection,"p011:fd:0010:batt_low",21,&addr);
}batt_timer++;
}
/* Send Alive*/
if(alive_timer >= TIME_ALIVE) {
            alive_timer = 0;
            if(alarm_flag == ALARM_NO) {
            simple_udp_sendto(&multicast_connection,"p00d:fd:0010:alive",18,&addr);         }
else if(alarm_flag == WAITING_STATE) {
            waiting_state++;
            simple_udp_sendto(&multicast_connection,"p011:fd:0010:waiting",20,&addr);
if(waiting_state >= waiting_flag) {
            alarm_flag = ALARM_NO;
            waiting_state = 0;
                                    }
                        }
            }
alive_timer++;
```

If the device, all the time sends the different states, two problems happen:

- Saturation Packets in the channel Wifi.
- Excessive consumption of energy.

So, is need eTimers for controlling the frequency of send this state information.

Hardware

Atmega128RFA1

The Atmega128RFA1 device is used by their characteristics and the microcontroller is chosen by the working group, remember this prototype is part of a one Project of the Fachhochschule Düsseldorf, the most main feature of the Atmega128RFA1 is:

Parameter Value	Max Data Rate (Mb/s):2
Flash (Kbytes):128 Kbytes	Antenna Diversity: Yes
Max. Operating Freq. (MHz):16 MHz	Power Output (dBm):3.5
TWI (I2C):1	Receiver Sensitivity (dBm):-100
UART:1	Receive Current Consumption (mA):16.6
Operating Voltage (Vcc):1.8 to 3.6	Transmit Current Consumption (mA):18.6
Timers:6	Link Budget (dBm):103.5
Frequency Band:2.4 GHz	CPU:8-bit AVR

Atmega128RFA1 is a powerful microcontroller, low power consumption, integrated wireless communication, therefore is a perfect chip for our prototype. Contiki can be implemented using wireless communication with one device, efficiently.
Let us remember that the goal is an affordable, low-power device, so the Atmel microcontroller is perfect for the goals we seek.

Windows detector

For control the opening of windows or doors, the system selected is "Magnet Reed Contact Set MRS" of the manufacturer ABB.

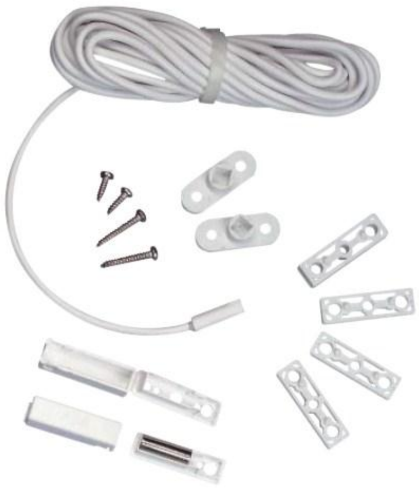


Figure 2: Magnet Reed Contact Set

The contact device is magnetically operated for a separate permanent magnet, the two units are mounted in parallel (for surface mounting) or end (for drilling). It's a simple but efficient and cheaper system.

RF circuit

When using an asymmetrical antenna, we need an impedance matching; this adaptation is achieved with the **Filter-Balun** of the Manufacturer Johanson Technology (6). This is optimized for Atmega128RFA1 [Figure 3]
It's possible make efficiency a cheapest antenna, using this filter and the Balun. The final set up is the next.

Filter.

We need filter the received signal to the antenna and the signal output from the microcontroller.

- **The received signal:**

The signals, need filtering because, only need the frequency of 2.4 GHz, because the work environment can have other emitting devices near this frequency, or any interference.

- **The emitted signal:**

The device, need to send the signal as clean as possible, without reflections and interferences that can generate the PCB, this interference appears because it's used a SMD components and use a manually welding process. When welding manually may be producing cold welding, this is an important problem when working in high frequency.

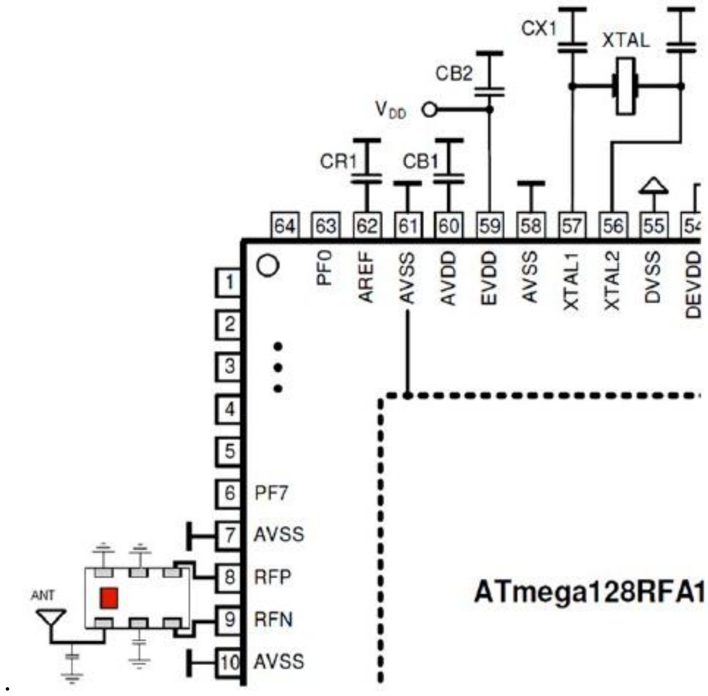


Figure 3: RF

Balun

This antenna not is symmetrical, for this reason is used the impedance matching circuit of the [Figure 4], L1 and L2 conform the matching circuit and C1 is for accurate the return loss.

This matching circuit is the recommended by the manufacturer (7), this matching circuit is important for obtain the correct return loss, and the bandwidth centered in 2.4 GHz [Figure 5]

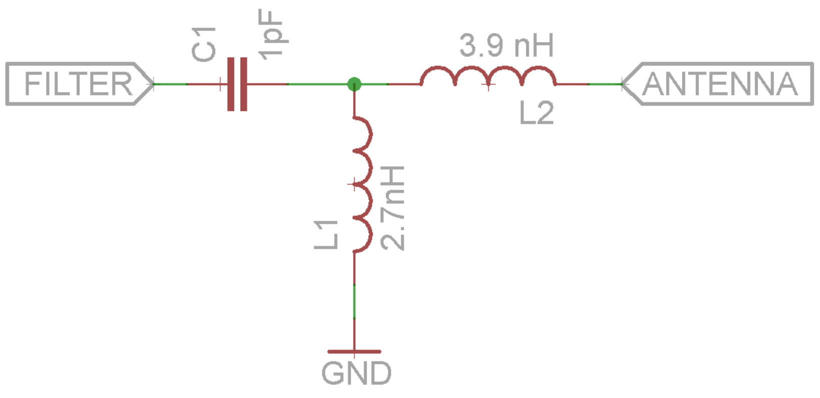


Figure 4: Matching circuit

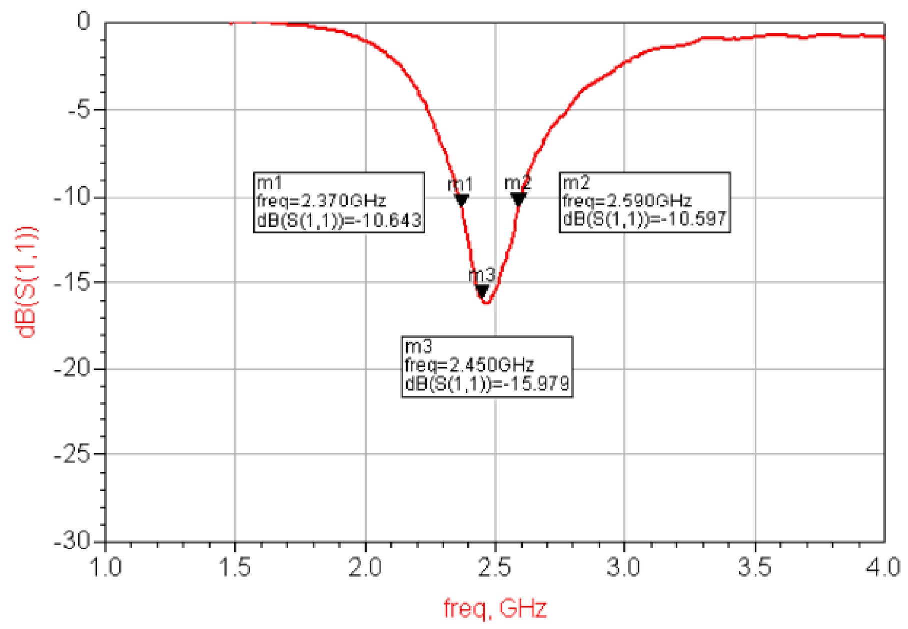


Figure 5: Return loss

Antenna

The antenna used in this design is manufactured by JOHANSON TECHNOLOGY; the model used is 2450AT18A100 - 2.45 GHz Antenna.

General Specifications			
Part Number	2450AT43B100	**Input Power**	2W max.
Frequency Range	2400 - 2500 Mhz	**Impedance**	50 Ω
Peak Gain	1.3 dBi typ. (XZ-V)	**Reel Quanity**	1,000
Average Gain	-0.5 dBi typ. (XZ-V)	**Operating Temperature**	-40 to +85°C
Return Loss	9.5 dB min.	**Recommended Storage Conditions (for unused product on T&R)**	+5 to +35°C, Humidity: 45-75%RH, 18 mos. Max

Typical Electrical Characteristics/Radiation Patterns (T=25oC)

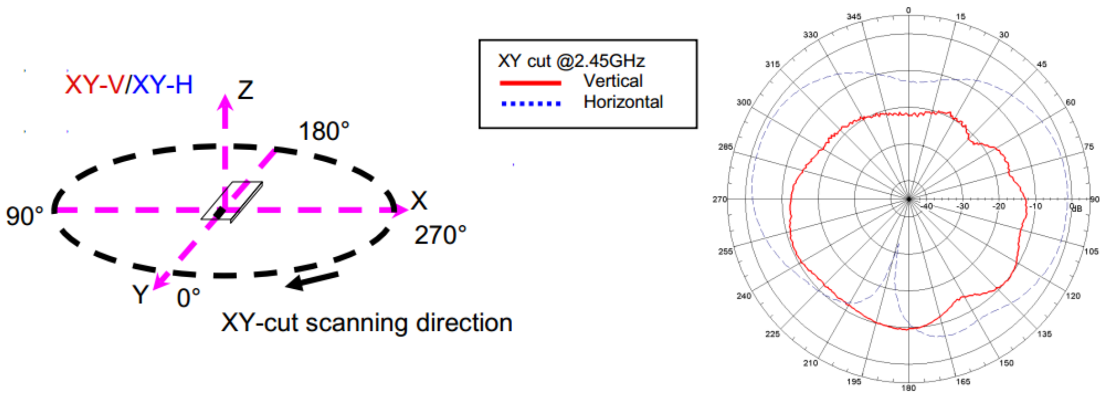


Figure 6: Radiation pattern

The prototype

Development targets

The objectives of the first version of the device are to be energy efficient and the size, using only the essential components and the smallest possible PCB design to install it in a window and not occupying space.

The total size should be no more than a 2xAA battery holder.

Explanation of each of the main parts of PCB and is distributed in the scheme of [Annex 3]

The PCB has the following parts:

- ATMEGA128RFA1 Circuit.
- Signal detection circuit.
- RF circuit.
- JTAG programmer
- Visual information and power.

ATMEGA128RFA1.

This set up; it's the circuit necessary for the normal operation of ATMEGA128RFA1. Using notes from the manufacturer Atmel (8), we use the next set up[1]:

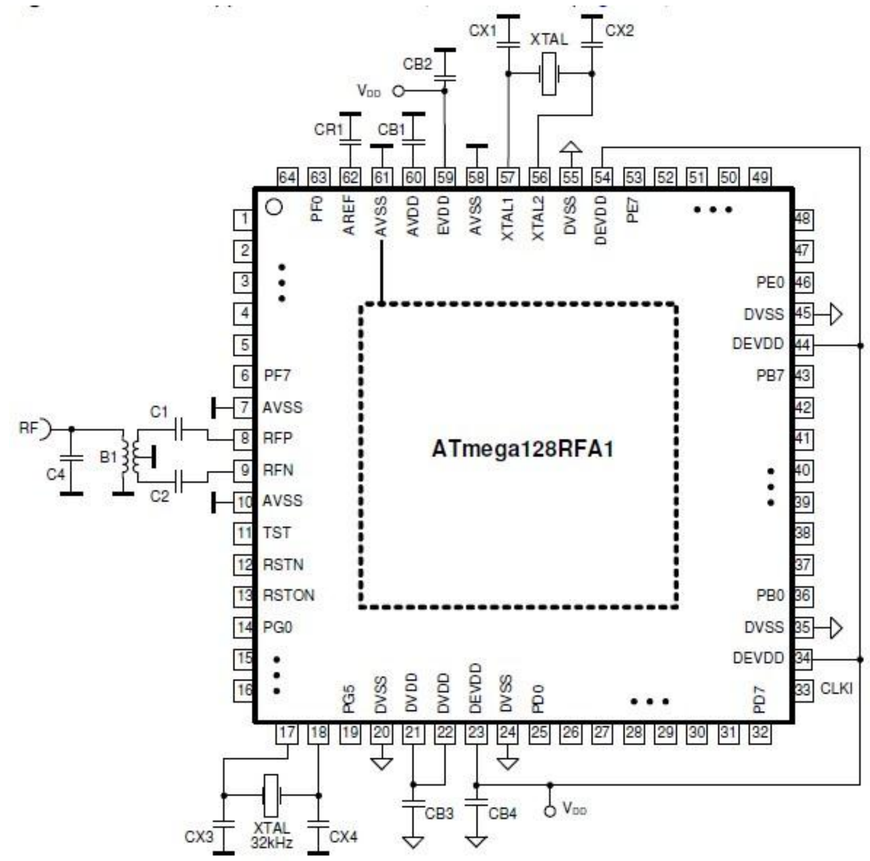
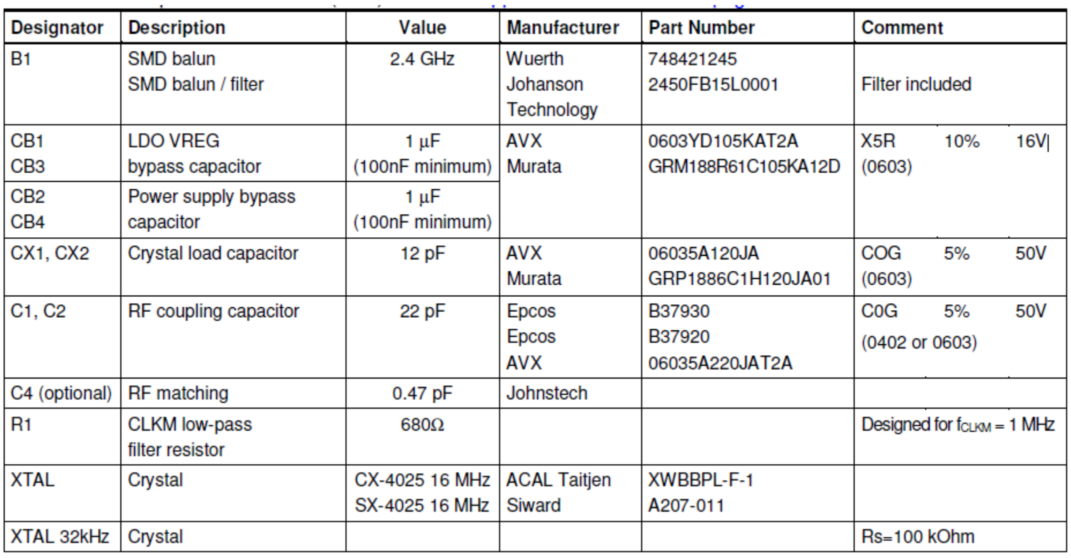


Figure 7: Basic set up

[1] Atmega Atmel128RFA1 preliminary, Application Circuits Basic Application Schematic, page 493

This set up ensures the correct operation of ATMEGA128RFA1 within the ranges guaranteed by the manufacturer.

For this we, use the following component values:

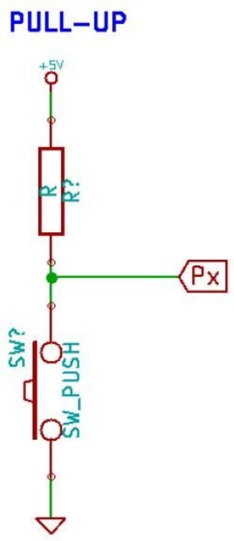


Designator	Description	Value	Manufacturer	Part Number	Comment
B1	SMD balun SMD balun / filter	2.4 GHz	Wuerth Johanson Technology	748421245 2450FB15L0001	 Filter included
CB1 CB3	LDO VREG bypass capacitor	1 µF (100nF minimum)	AVX Murata	0603YD105KAT2A GRM188R61C105KA12D	X5R 10% 16V| (0603)
CB2 CB4	Power supply bypass capacitor	1 µF (100nF minimum)			
CX1, CX2	Crystal load capacitor	12 pF	AVX Murata	06035A120JA GRP1886C1H120JA01	COG 5% 50V (0603)
C1, C2	RF coupling capacitor	22 pF	Epcos Epcos AVX	B37930 B37920 06035A220JAT2A	C0G 5% 50V (0402 or 0603)
C4 (optional)	RF matching	0.47 pF	Johnstech		
R1	CLKM low-pass filter resistor	680Ω			Designed for f_{CLKM} = 1 MHz
XTAL	Crystal	CX-4025 16 MHz SX-4025 16 MHz	ACAL Taitjen Siward	XWBBPL-F-1 A207-011	
XTAL 32kHz	Crystal				Rs=100 kOhm

Figure 8: Components and values

Signal detection circuit.

It's where we integrate the Magnet Reed Contact, as the magnetic sensor functions as a switch; the choice is to do the following circuit.



This circuit uses a pull-up resistor of port entries; this resistance is enabled by software. The external resistance is to prevent a short circuit.

The purpose is that the window be closed no energy is consumed and energy is consumed only when the window is open.

This setup increases the autonomy of the device, by not having to be consuming energy continuously and only do so at the time of the detection.

Figure 9: Signal detection circuit

RF

The RF part is responsible for communication with the central node; include the Balun-filter and antenna. As shown in the footprint, is located in a corner of the PCB to guarantee that the radiation pattern of the antenna is not distorted by the ground plane of the device, thus the maximum possible range is obtained.

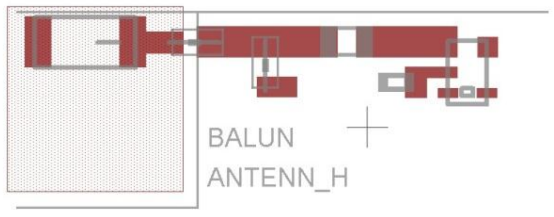


Figure 10: Balun footprint

Is one of trouble spots of the assembly, because the whole set is welded manually and to be working at 2.4 GHz, it is very easy to cause interferences, so we need clean weld.

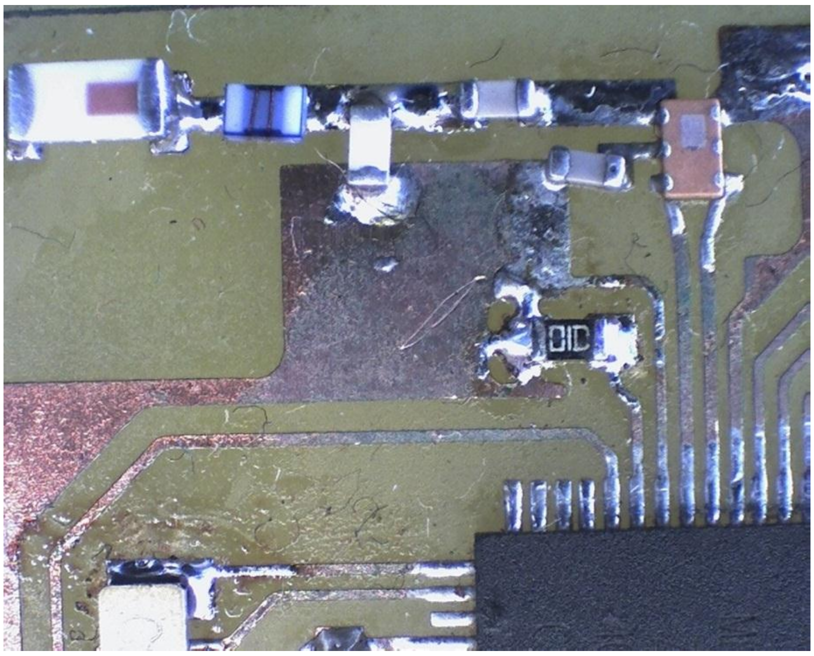


Figure 11: Balun detail at x10

JTAG programmer

The microcontroller it's programming using a JTAG interface, the JTAG interface can be used for:

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip debugging

But the debugging, only its possible without Contiki, it's impossible run the OS in real time, therefore, the first developing without Contiki and after embedded the OS.

For the programming use the AVR JTAGICE mkII of the manufacture Atmega

Visual information and power

For visual information from the correct function of the device during the design incorporates a LED, only have the information of the magnetic sensor is close or open.

When finishes design the schematic, the next step is routing the wires. For this work it's use the manually routing, the result is the next footprint.

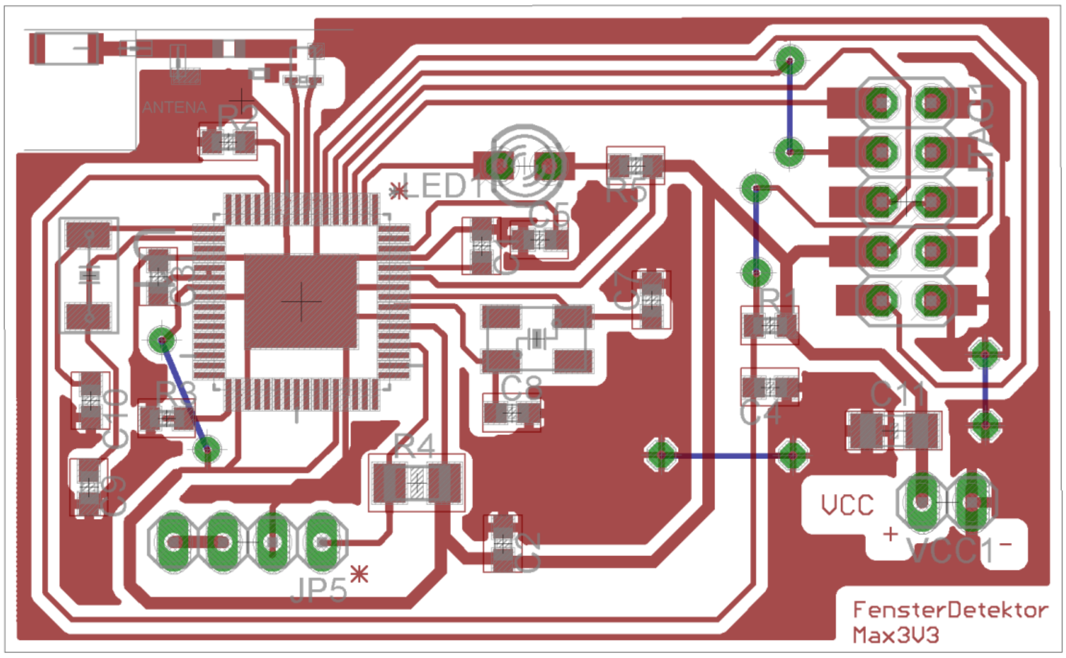


Figure 12: Device Footprint

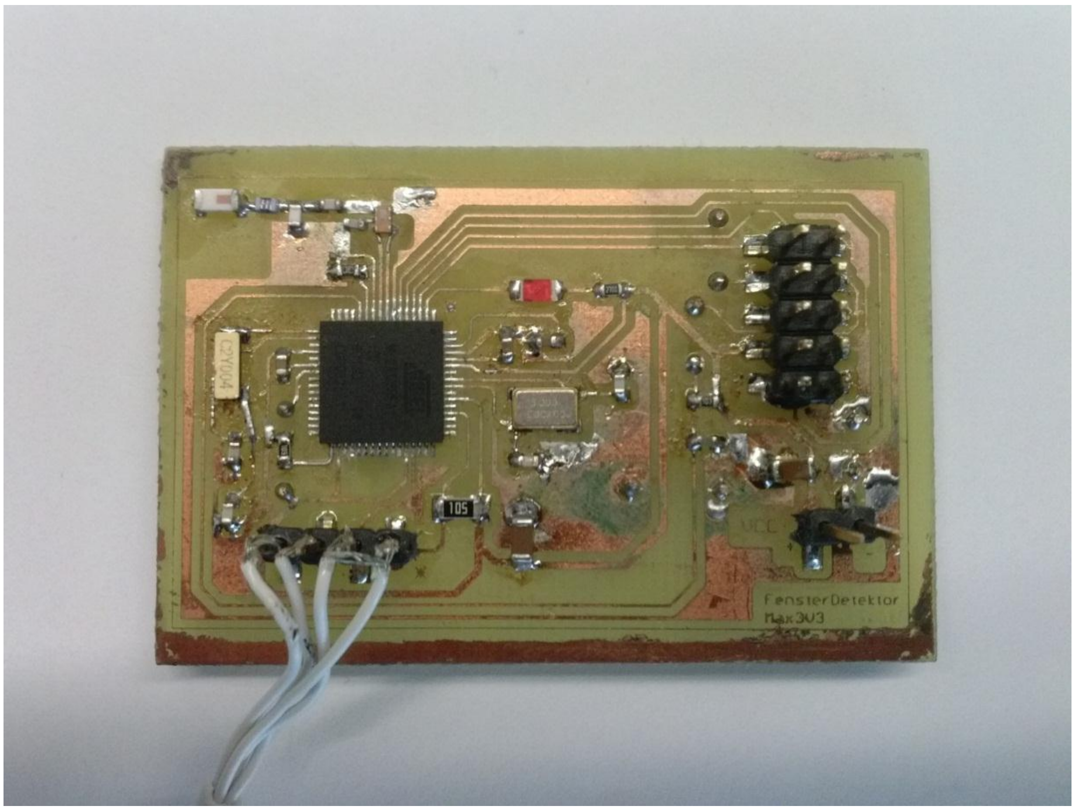


Figure 13: Prototype

Simulation

To check the correct operation of the entire device, perform the following steps.

1. Basic functions
2. Adding Contiki
3. eTimers and functions
4. Optimize the power consumption

First, the code is made without the use of Contiki, I only use a C code that does the basic functions are performed. After the OS code was introduced and tested that the code worked correctly together, this part is difficult because, you cannot use the debug mode in the Atmel Studio.

Need first make a functionally template of the OS Contiki working correctly, this template it's an extract of the different documents of the Contiki Community and Contiki developers and I need modified for our objectives.

Lastly, the timers were introduced to manage the different actions taken by the device and optimize energy consumption. This parts it's one of the most important because when less power consumption more time of life for our device without need replace de battery.

Then, it's needed optimize the power consumption, so it's used the next command power_all_disable() .Contiki enable all the peripherals of Atmega128RFA (ADC,timers,WART….), this peripherals consume extra energy around of 0.27mA. When use de Green Mode, the device only consume 0.07 mA, this consume, guarantees a 15000 hours of autonomy, adding the consumption of the times of checking the device (Alive),the life span of the device is around one year of autonomy using a standard battery AA

For the simulation of the whole device, one sniffer network packet is used; the software used is Wireshark (9).

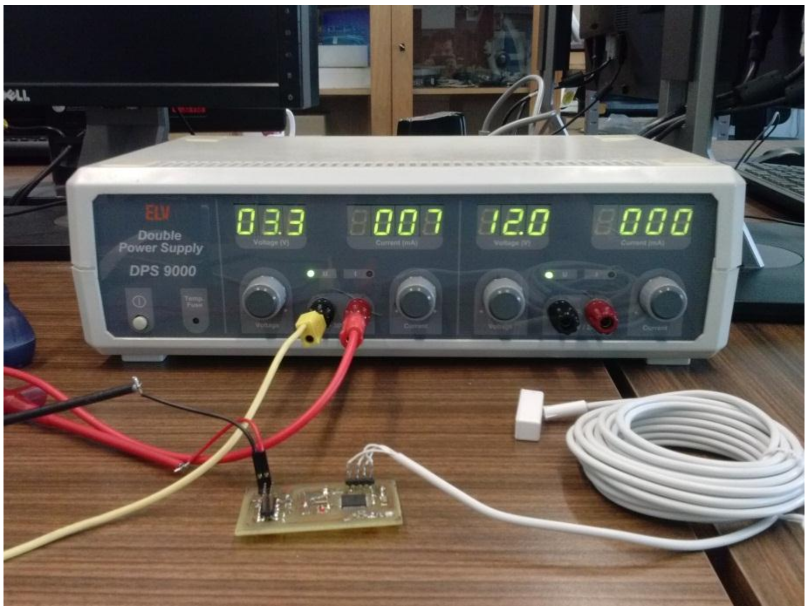
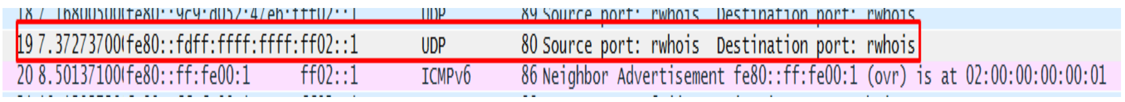


Figure 14: Simulation Setup

Wireshark Simulation

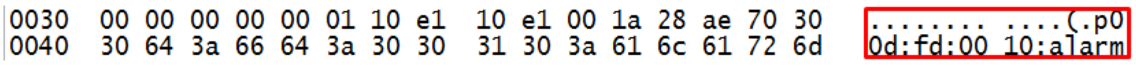
The test is to connect the device and use a network sniffer. We connect the laptop to the router, the router works as a central node. Now start monitoring the network, thus capture all packets you send or receive to the central node. The process followed it’s the next:

- First, is simulated that the window it's opened.



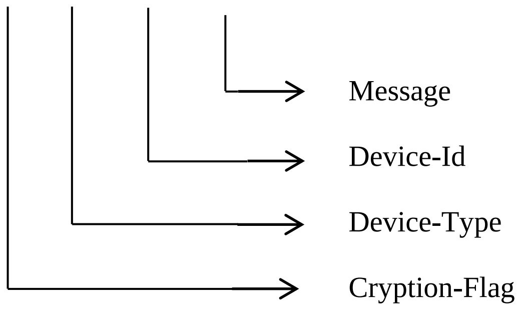
```
18 7.16800500fe80::9c9:d052:47eb:fff02::1     UDP     89 Source port: rwhois  Destination port: rwhois
19 7.37273700(fe80::fdff:ffff:ffff:ff02::1    UDP     80 Source port: rwhois  Destination port: rwhois
20 8.50137100(fe80::ff:fe00:1      ff02::1    ICMPv6  86 Neighbor Advertisement fe80::ff:fe00:1 (ovr) is at 02:00:00:00:00:01
```

The packet sent for our device, contains the information of the next image, this information means



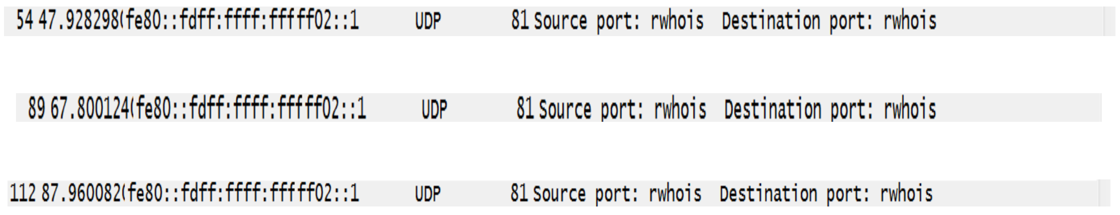
```
0030   00 00 00 00 00 01 10 e1   10 e1 00 1a 28 ae 70 30   ........ ....(.p0
0040   30 64 3a 66 64 3a 30 30   31 30 3a 61 6c 61 72 6d   0d:fd:00 10:alarm
```

p00d : fd : 00 10 : alarm

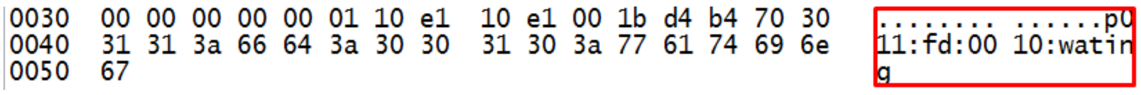


- The device sent correctly the alarm packet.

The next part is close the window and checking that the device waiting the time correctly for sending the Alive packet. For sending one Alive packet Its need waiting 3 waiting_state.

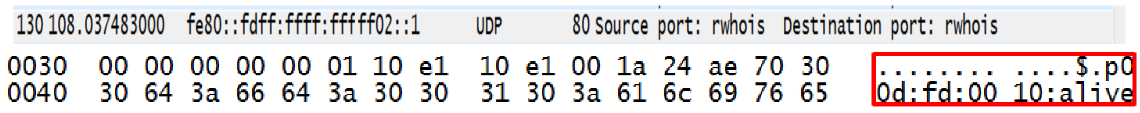


```
54 47.928298(fe80::fdff:ffff:ffff02::1     UDP     81 Source port: rwhois  Destination port: rwhois

89 67.800124(fe80::fdff:ffff:ffff02::1     UDP     81 Source port: rwhois  Destination port: rwhois

112 87.960082(fe80::fdff:ffff:ffff02::1    UDP     81 Source port: rwhois  Destination port: rwhois
```



```
0030   00 00 00 00 00 01 10 e1   10 e1 00 1b d4 b4 70 30   ........ ......p0
0040   31 31 3a 66 64 3a 30 30   31 30 3a 77 61 74 69 6e   11:fd:00 10:watin
0050   67                                                  g
```

After waiting, the time set in the wating_state variable, the device sends an "Alive"



```
130 108.037483000   fe80::fdff:ffff:ffff02::1     UDP     80 Source port: rwhois  Destination port: rwhois
0030   00 00 00 00 00 01 10 e1   10 e1 00 1a 24 ae 70 30   ........ ....$.p0
0040   30 64 3a 66 64 3a 30 30   31 30 3a 61 6c 69 76 65   0d:fd:00 10:alive
```

Conclusions

During the whole design process, I have defined some essential objectives, first the energy efficiency because the device is a wireless sensor, they having a finite energy source, in this case the power supply is a 2xAA battery and second, the importance of their size as it is intended to be as discrete as possible.

Also, I had to use a default hardware and software, since the sensor had to work on a project already started, I having to adapt the design ideas to the working environment.

After studying the Hardware and Software, I proceeded to prototype design and manufacture in the laboratories of the Fachhochschule; I have completed the process with a verification of correct operation and compliance with the main goals.

We may conclude that using the microcontroller ATMEGA128RFA1 with Contiki, it's possible to develop wireless sensors that are very energy efficient at low cost.

Problems and Solutions

Software

The little documentation of Contiki:

Contiki has a large development community but don’t have a written documentation, books or courses, compared to other operating systems (TinyOs), so the time required for compression have to be superior.

The solution was the learning, using the community of developers and studying the codes used in other device, and finally extract a templates for using the most common functions in Contiki (communication via IPv6 and use the timers)

The debug process:

Contiki is an OS introduced in the ATMEGA128RFA1; he does not have the option of debugging in execution mode, i.e. in real time during the normal function. This detail, increase the difficult to find bugs in the code, especially in the use of timers.

The only form is starting in the basic functions and finish uses all characteristics of Contiki, Wifi connection, timers and multiples process.

Hardware

Manufacture process

The components used in the device having a SMD encapsulated, for this reason the PCB had many critical areas for the manufacture process, pads close together and small wires.

When the manufacturing process is performed, even using professional techniques for make the PCB, I had to repeat the process several times because appear errors in the manufacture, cut tracks or pads together.

Assembly process

When weld SMD devices manually, I have the following problems.

ATMEGA128RFA1:

The device, don’t have an external pins, the atmega128RFA1, it's made for welding using wave welder or reflow welded, for this reason when I use a manual welding I need a special caution in the position of the device over de pads of the PCB

The default footprint of the device was modified and I have using a footprint with pads more extended. For facilitating the manual welding pads, I used magnifiers and professional smd welding station.

Balun

The RF part it’s a critical are the circuit, because operate at higher frequencies 2.4 GHz, welding difficulty of the coils to weld, this caused the RF part will not work.

I having problems in the welding of the coils, appears a cold weld rings, for this reason the RF part will not work.

Delicate welds were performed using Flux and change the coil for one of other type of encapsulation.

Others

Closed Lab:

For a month the laboratory manufacturing PCB Fachhochschule was closed.

Annexes

1. Contiki with Atmel Studio

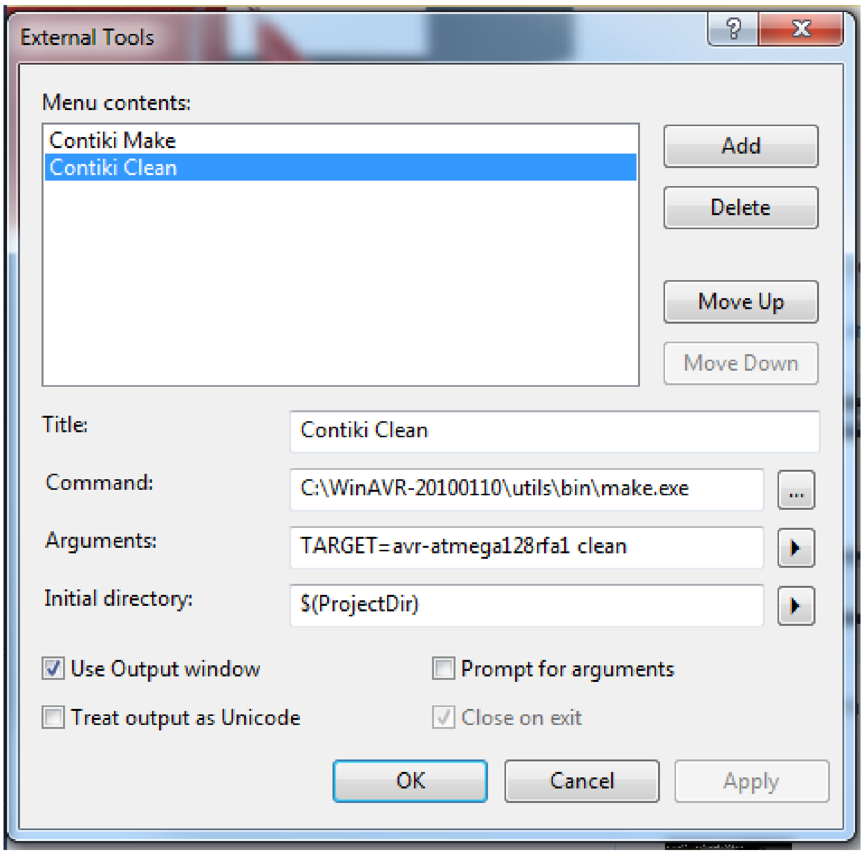
For use Contiki with Atmel Studio it's needed follow the next steps:

1. Download Contiki[2], and copy in a directory, for example c:/Contiki.
2. Download and install the WinAVR[3], is a suite of executable, open source software development tools for the Atmel AVR series.

Then, go to the Atmel Studio and add the external tools necessary for generate the files to programming the microcontroller.

Atmel Studio→Tools→External Tools

The first external tools is Contiki Clean:


External Tools

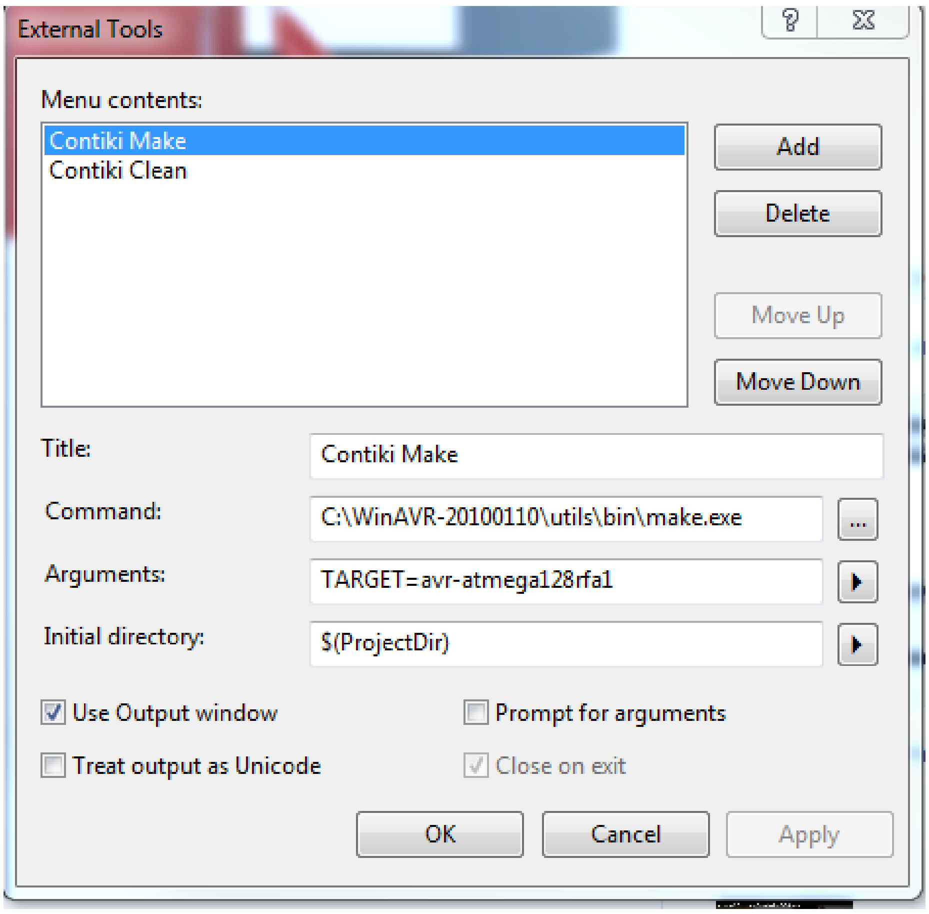
Menu contents:
Contiki Make
Contiki Clean

Title: Contiki Clean
Command: C:\WinAVR-20100110\utils\bin\make.exe
Arguments: TARGET=avr-atmega128rfa1 clean
Initial directory: $(ProjectDir)

[✓] Use Output window
[ ] Prompt for arguments
[ ] Treat output as Unicode
[✓] Close on exit

[2] http://www.contiki-os.org/
[3] http://winavr.sourceforge.net/

The second external tools are Contiki Make:



Menu contents:
- Contiki Make
- Contiki Clean

Title: Contiki Make

Command: C:\WinAVR-20100110\utils\bin\make.exe

Arguments: TARGET=avr-atmega128rfa1

Initial directory: $(ProjectDir)

[✓] Use Output window
[ ] Prompt for arguments
[ ] Treat output as Unicode
[✓] Close on exit

Be careful to enter the parameters.

Every times need compile the project for programming the device, need run the two steps in this order.

- Contiki Clean
- Contiki Make

When finish the make process, a one file is generated, this file its name Project_Name.avr-atmega128rfa1. This is the file for programing the Atmega128RFA1.

2. Code windows state sensor

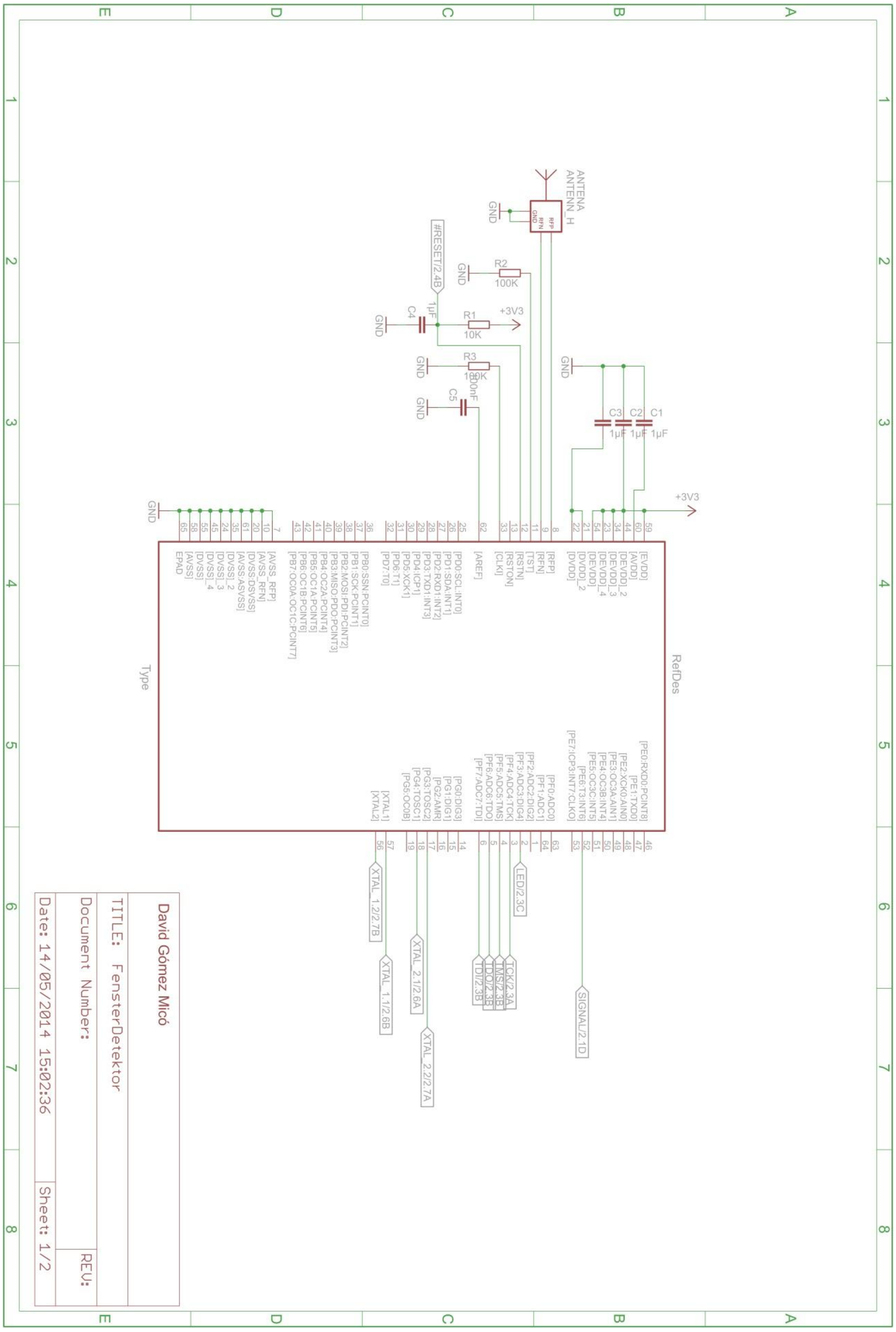
```
/*
*Author: David Gómez Micó
*/

#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include "temp.h"
#include <avr/sleep.h>

#ifdef USE_SLEEP
#include "sleep.h"
#endif

/*Definition für vereinfachten Zugriff auf µIP-Datenpuffer*/
/*Definition for simplified access to ?IP data buffer*/
#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])

/* Intervalle definieren */
/* Definition Time Interval*/
#define TIME_PERIODE 5                  // Sekunden einer Schlafperiode
                                        // Seconds for enter in Sleepmode
#define TIME_ALIVE              4       // Anzahl der Schlafperioden bis "alive"
                                        // Time for sending Alive
#define TIME_BATT               10      // Anzahl der Schlafperioden bis mögl. "batt_low"ry
                                        // Timer for sending Battery low

/*GREENMODE */
/*Low power consuption*/
power_all_disable();

/* Vereinfachende Definitionen */
/* Simple Defines*/
#define ALARM_NO        0
#define ALARM                   1
#define WAITING_STATE           2

/* Funktionsprototypen */
/* Prototype Functions */
static void tcpip_handler(void);        //Auswertung von eingehenden Daten
                                        //Evaluation of incoming data
/* Struct zur Speicherung von Gerätedaten und Ports */
/* Struct for storing device data and ports */
        typedef struct {
        char cryption;      //Cryption-Flag
        char *type;         //Device-Type
        char id[5];         //Device-Id
        int in;             //Port für eingehende Daten
        int out;            //Port für ausgehende Daten
        } device_t;
```

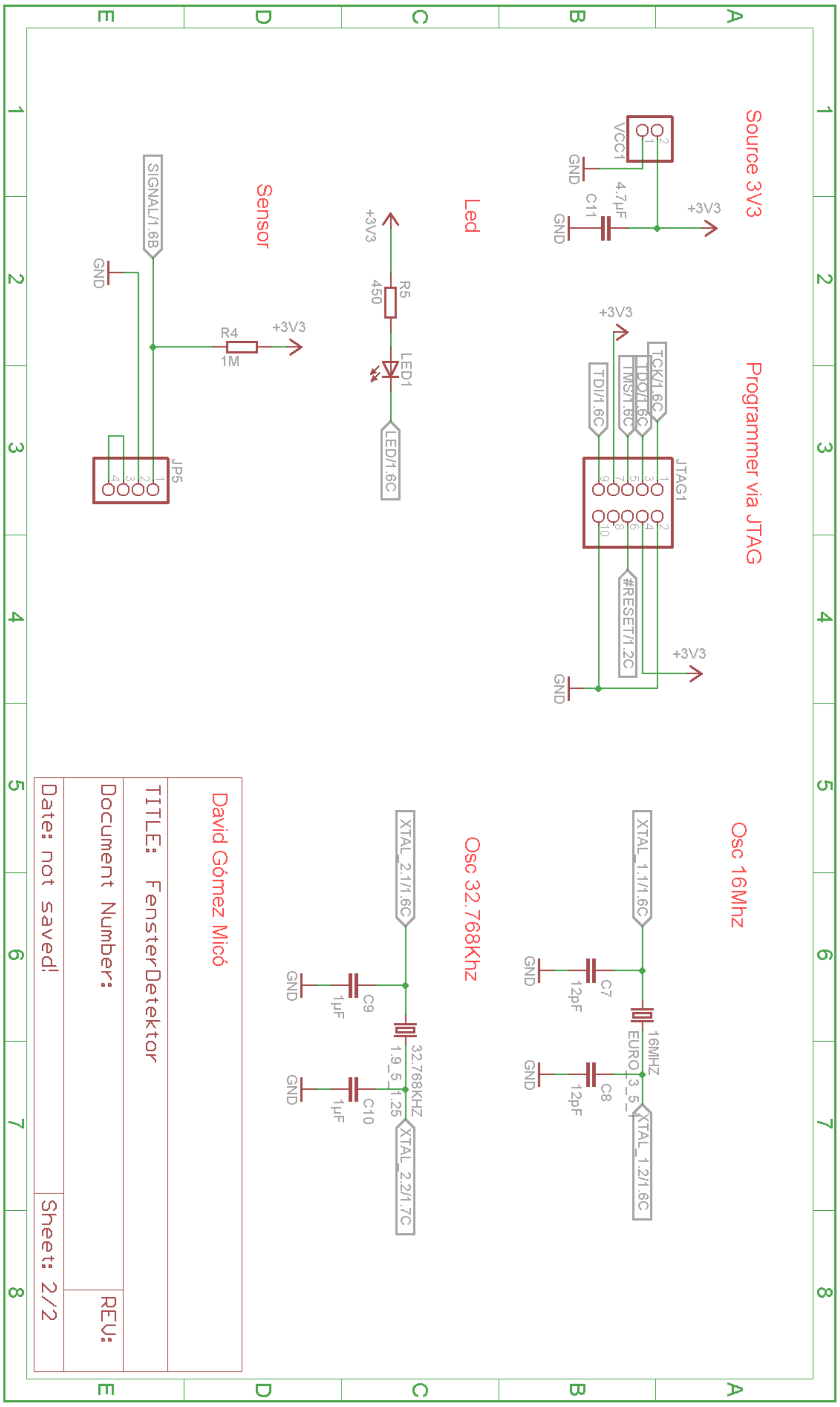
```
/* UDP-Ports und Gerätedaten definieren */
/* UDP-Ports define device */
static device_t device = {
            'p',        //Cryption-Flag
            "fd",       //Device-Type
            "0010", //Device-Id
            4321,     //Port für eingehende Daten
            4321};   //Port für ausgehende Daten

/* Struct für die UDP-Verbindung des Servers */
/* Struct for the UDP connection of the server */
static struct uip_udp_conn *server_conn;

/* Prozesse der Anwendung */
/* Processes of the application */
PROCESS(udp_server_process, "UDP server process");
PROCESS(Fensterdetektor, "Main Process");

/* Prozesse automatisch starten */
/* Automatic Process start */
AUTOSTART_PROCESSES(&Fensterdetektor);

/* ---------------------------------------------------------------------------
* udp_server_process: Nimmt UDP-Pakete entgegen.
* udp_server_process: Accepts UDP packets.
* ---------------------------------------------------------------------------
*/
PROCESS_THREAD(udp_server_process, ev, data) {

/* Beginn der Prozessausführung */
/* Beginning of the process execution */
PROCESS_BEGIN();

/* Stromsparfunktionalität initialisieren */
/* Initialize power saving functionality */
SLEEP_INIT();
CPU_SLEEP_DISALLOW();
RF_SLEEP_DISALLOW();

/* Verbindung für Datenempfang erstellen */
/* Create a connection to receive data */
server_conn = udp_new(NULL, UIP_HTONS(0), NULL);
udp_bind(server_conn, UIP_HTONS(device.in));
while(1) {
            /* Kontrolle abgeben und auf Event warten */
            /* Make check and wait for event */
            PROCESS_YIELD();
            if(ev == tcpip_event) {
             tcpip_handler();
            }
}
PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```
/* ---------------------------------------------------------------------------
* alive_process
* ---------------------------------------------------------------------------
*/
PROCESS_THREAD(Fensterdetektor, ev, data) {

        DDRF = 0xff; //Led
        DDRE = 0x00; //Sensor
        PORTE = 0xff; //Pull up On

        /* Struct und Adressvariable für den UDP-Broadcast */
        /* Structure and Variable for the UDP-Broadcast */
        static struct simple_udp_connection multicast_connection;
        static uip_ipaddr_t addr;

/* Beginn der Prozessausführung */
/* Beginning of the process execution */
PROCESS_BEGIN();
/* Stromsparfunktionalität initialisieren */
/* Power save function*/
SLEEP_INIT();
CPU_SLEEP_DISALLOW();
RF_SLEEP_DISALLOW();

/* Variablen deklarieren und initialisieren */
/* Variables */
        static struct etimer timer;
        static struct etimer timer_peri;
        static uint8_t alive_timer = TIME_ALIVE;
        static uint8_t batt_timer = TIME_BATT;
        static uint8_t alarm_flag = ALARM_NO;
        static uint8_t waiting_state = 0;

/* Battereieüberwachung einstellen: 2,45V für Stabilität bei 8MHz */
/* Battery configuration for stability to 8Mhz*/
batt_configure(V2_45);

/* UDP-Verbindung registrieren */
/* Regist UDP*/
simple_udp_register(&multicast_connection, device.out,NULL, device.out,NULL);

etimer_set(&timer, (CLOCK_SECOND * 3));
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));

/* Multicast als Empfängeradresse */
/* Multicast address as the recipient */
uip_create_linklocal_allnodes_mcast(&addr);

/* Periodischen Timer setzen */
/* Set the periodic timer */
etimer_set(&timer_peri, CLOCK_SECOND * TIME_PERIODE);
while(1) {
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer_peri));
        if (PINE & (1<<PINE6)) //If PF0 = 0 trigger condition, open window.{
                PORTF &= ~(1 << 3);// Led On
                alarm_flag = ALARM;
                alive_timer = 0;
                waiting_state = 0;
                simple_udp_sendto(&multicast_connection, "p00d:fd:0010:alarm", 18, &addr);
        }else{
            PORTF |= (1 << 3);
            if(alarm_flag == ALARM) {
            alarm_flag = WAITING_STATE;}
            }//else
            /* Batterie testen? */
            /* Battery test? */
            if(batt_timer >= TIME_BATT) {
                        if(batt_check_threshold() == BELOW_THRESHOLD) {
                        batt_timer = 0;
                        simple_udp_sendto(&multicast_connection, "p011:fd:0010:batt_low", 21, &addr);}
                        batt_timer++;
                        }

/* "alive" senden? */
/* Send Alive*/
if(alive_timer >= TIME_ALIVE) {
alive_timer = 0;
if(alarm_flag == ALARM_NO) {
            simple_udp_sendto(&multicast_connection, "p00d:fd:0010:alive", 18, &addr);}
            else if(alarm_flag == WAITING_STATE) {
                        waiting_state++;
                        simple_udp_sendto(&multicast_connection, "p011:fd:0010:waiting", 20, &addr);
                                    if(waiting_state >= 4) {
                                    alarm_flag = ALARM_NO;
                                    waiting_state = 0;}
                                    }}alive_timer++;

            /* Schlafen der RF-Einheit wieder erlauben */
            /* Sleeping allow the RF unit again */
                        if(!RF_SLEEP_ALLOWED()) {
                                    etimer_set(&timer, CLOCK_SECOND / 2);
                                    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
                                    RF_SLEEP_ALLOW();}

            /* Schlafen der CPU wieder erlauben */
            /* Allow the CPU to sleep again */
                        if(!CPU_SLEEP_ALLOWED()) {
                                    etimer_set(&timer, CLOCK_SECOND / 2);
                                    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
                                    CPU_SLEEP_ALLOW();}

/* Auf periodischen Timer warten und diesen neu starten */
/* Wait for periodic timer and restart it */
etimer_reset(&timer_peri);
  }
  /* Ende der Prozessausführung */
  /* End of the process execution */
```

```
 PROCESS_END();
}
/*---------------------------------------------------------------------------*/

/* ---------------------------------------------------------------------------
 * tcpip_handler: Wertet Daten aus empfangenem UDP-Paket aus.
 * tcpip_handler: Evaluates data from the received UDP packet.
 * ---------------------------------------------------------------------------
 */
static void tcpip_handler(void) {
static uip_ipaddr_t ipaddr;//Adressvariable
static char cmdLine[100];              //Variable zur Erzeugung "protokollkonformer" Daten
                                       // Variable data to produce "protocol-compliant"
  uint8_t length;                      //Länge der Daten
                                       //length data
/* Möglichen Befehl nach Protokoll zusammensetzen */
/* Composed possible instruction on the protocol */
        static char command[10] = "ping";
        length = 6 + strlen(device.type) + strlen(command);
        sprintf(cmdLine, "%c%03x:%s:%s:%s", device.cryption, length, device.type, device.id,
command);

        /* Neue Daten in µIP-Stack? */
        /* New data in ?IP stack? */
  if(uip_newdata()) {
        /* String (mit '0') terminieren */
        /* Terminate string ('0') */
        ((char *)uip_appdata)[uip_datalen()] = 0;
        /* Eingehende Daten auf Befehl überprüfen */
        /* Check incoming data on command */
        if(strcmp(uip_appdata, cmdLine) == 0) {
                /* Verbindung auf "ausgehend" einstellen (Multicast) */
                /* Setting the connection to "outgoing" (multicast) */
                        uip_ip6addr(&ipaddr,0xff02,0,0,0,0,0,0,0x0001);
                        uip_ipaddr_copy(&server_conn->ripaddr, &ipaddr);
                        server_conn->rport = UIP_HTONS(device.out);

                        /* Antwort senden */
                        /* Send reply */
                        uip_udp_packet_send(server_conn,  "p00c:fd:0010:pong", 17);

                        /* Verbindung auf Empfang zurücksetzen */
                        /* Reset connection on reception */
                        memset(&server_conn->ripaddr, 0, sizeof(server_conn->ripaddr));
                        memset(&server_conn->rport, 0, sizeof(server_conn->rport));
                }
  }
}
/*---------------------------------------------------------------------------*/
```

3. Schematics




Source 3V3
Programmer via JTAG
Osc 16Mhz
Led
Osc 32.768Khz
Sensor
VCC1
GND
+3V3
4.7µF
C11
JTAG1
TCK/1.6C
TDO/1.6C
TMS/1.6C
TDI/1.6C
#RESET/1.2C
R5
450
LED1
LED/1.6C
R4
1M
SIGNAL/1.6B
JP5
XTAL_1.1/1.6C
XTAL_1.2/1.6C
C7
C8
12pF
16MHZ
EURO_3_5_4
XTAL_2.1/1.6C
XTAL_2.2/1.7C
C9
C10
1µF
32.768KHZ
1.9_5_1.25
David Gómez Micó
TITLE: FensterDetektor
Document Number:
REV:
Date: not saved!
Sheet: 2/2

List of figures

Bibliography

1. [Online]. Available from: http://www.cadsoftusa.com/.

2. [Online]. Available from: http://www.atmel.com/microsite/atmel_studio6/.

3. Adam Dunkels BG. Contiki - a Lightweight and Flexible Operating System for Tiny Networked..

4. Contiki-os. [Online]. Available from: http://contiki.sourceforge.net/docs/2.6/a01667.html.

5. Reusing T. Comparison of Operating Systems,Seminar: Sensorknoten - Betrieb, Netze & Anwendungen SS2012; 2012.

6. Johanson Technology. [Online]. Available from: http://www.johansontechnology.com/datasheets/balun-filter/2450FB15L0001.pdf.

7. Johanson Technology. [Online]. Available from: http://www.johansontechnology.com/datasheets/antennas/2450AT18A100.pdf.

8. Atmel. [Online].; 2014 [cited 2014. Available from: http://www.atmel.com/devices/atmega128rfa1.aspx.

9. Wireshark. [Online]. Available from: http://www.wireshark.org/.