Sensing

Introduction

The goal of this year's wireless sensor networks lab (WSN) is to build a wireless network of sensor nodes, equipped with sensors to measure the humidity, temperature and carbon dioxide (CO2) in an exclusive environment – the solar house (surPLUShome). The selected device is the Zolertia Z1 sensor node. It is suitable for wireless sensor networks because its micro-controller TI MSP430 is especially build for low power applications. Furthermore it has a wireless module compliant with IEEE 802.15.4. The objective of the sensing project role is to equip this node with missing sensors and integrate them into the Contiki operating system. The Z1 comes with a temperature sensor and a digital accelerometer.

Identification and purchase of extra sensors

As already mentioned there is a need to measure humidity, temperature and CO2. Since the Z1 is equipped with a temperature sensor, only humidity and CO2 sensors had to be purchased. In the beginning of the project was also a light sensor desired. So there is additional the opportunity to easily use this sensor (a driver is provided). The most simple way to capture values from the analog world (everything needed is continuous) is to use an ADC (analog-digital converter). Hence we have chosen a humidity sensor with a linear output voltage – the Honeywell HIH-5030. CO2 sensors are very expensive and there is often a necessity to calibrate them. Therefore we bought a complete sensor station containing temperature, humidity and CO2 sensors – Voltcraft CO-50. Two nodes are connected to them, the other uses the internal temperature sensor and the Honeywell humidity sensor.

Installation (connection/soldering)

The Z1 provides a couple of connectivities for digital buses, I/Os for ADCs and general purposes. Overall are seven AD channels available. The humidity is soldered to AD channel number one with a 65 kOhm resistor. The sensor station has a output connection using the I2C protocol. We have measured a clock with two kHz and data packets on another pin with an oscilloscope. Unfortunately the I2C port on the Z1 is already used by the two internal sensors. So we build an intermediate system with a pic micro-controller. This controller samples the data packets and stores the sensor values internally. Since we have not known before how the values are presented and in which packet, we have logged a couple of data and tried to find a correlation between them and the displayed values on the station. We have found this correlations, but the sampled values are not the same as the displayed, therefore we have added computations to get the exact results in a simple manner. These final values are then transmitted via UART to the node.

Development of drivers

The drivers in Contiki are divided into two different categories. There are driver for different micro-controller like UART and driver for a special platform like the temperature sensor. As we added external sensors, we developed only driver that are suitable for the Z1 platform. They can be found in Contiki under /platform/z1/dev. New developed driver are added to the Makefile, too. The Sky node is similar to the Z1 node because both use the same micro-controller. So I have copied the configuration routine for ADC and adapted it to the Z1 platform. These routine files are labeled z1-sensors.{c|h} and contains methods to configure a special ADC channel and to get the current status of it. The developed driver just defines the channel and some other parameter and offers the sensor methods to configure and to read the captured value. This is done for the humidity and the light sensor. The more difficult part is to get the sensed values from the sensor station. As mentioned above we sampled these values with an intermediate system. This micro-controller has a short routine that triggers data on a positive clock edge. We have seen that nine different packets are sent because nine equal bytes appeared periodically. These header are followed by a couple of another bytes of which two seemed to be data. Then we printed them out on a console and tried to find a

correlation by increasing the temperature, CO2 or humidity values. After many measured and logged data we have found this correlation and adjusted the code to store these exact values. We have used then the simplest way to provide these data to the Z1 node by sending them via UART. The UART driver can be found under /cpu/msp430/dev/uart1x.c. Unfortunately this driver has been copied by the original developers from the sky node without adjusting them. So I corrected it to give us the possibility to receive packets. The packets sent by the pic controller invokes an interrupt that is handled by a developed method. This method in /platform/z1/dev/sensorstation.{c|h} stores the packets internally. Each transmission is separated by a special byte, followed by the values for CO2, humidity and temperature. Since this complete step is not very quick, all values are stored in a struct and are replaced with new arriving data. The sensor station driver provides a method to configure the UART and to return this sensor struct. All values are not needed in such a short interval that this steps suffice related to speed.

Calibration

The datasheets for additional sensors gives linear correlations for the measured voltage to the current values. But the conversion is not very accurate. The sensor station itself is already calibrated, so I have used its current values to calibrate the other additional sensors. This calibration is done by simple multiplication or shifting the ADC values. For the lack of being able to handle floating point numbers, I converted the sensed values in such a way that only integer values are used. The temperature for example leads from 24.5 degree to 2450. Besides this integer value is only one value to transmit to other nodes, instead of using two values for comma separated data.

Integration of sensing into TikiDB

TikiDB has one part that is responsible for reading sensor data, the queryresolver. There are already a couple of sensors used by the Sky mote before. So I have replaced one light sensor with the new CO2 sensor method and reused the given temperature and humidity methods. The queryresolver itself resolves the incoming queries by us of queryresolver-util. In the queryresolver sensors are started by calling their calibration methods. The data scheme can be found in the header file where I have added the CO2 sensor, too. For every query the resolver uses a switch-case selection to call the responsible sensor method. This methods are inside the queryresolver-util file and invokes depending on what node Contiki is running, the sensor station read function or just the described ADC function. The returned sensor values then are stored in the corresponding column of TikiDB.

Problems and challenges

To create and use the ADC part is not very difficult because it is done by simply configure the converter and it runs without use of Contiki in the background. The values are stored in a defined memory and can be collected every time. The more difficult part is the sensor station. As already explained we have spent much time with finding a correlation between displayed and triggered data. After this has succeed we have tried to find a way to send data to the Z1 node. The first trial was reusing the I2C. The Z1 acts as a Master for the given temperature and humidity sensor, which are digital I2C slave sensors. Unfortunately the trial interrupted or disturbed by the other sensors in such a way that their values occurred instead the sensor station ones. Further both micro-controller work in another voltage domain so that the pic detected incoming messages but the Z1 not. Thereafter we decided to use the UART, as it is very simply, but we have got some troubles, too. The arriving of bytes invokes an interrupt, but only one byte has been arrived in a sending interval. We have then detected that an overflow error occurs every second time. Normally the UART is as quick as the sending part. Especially is this case, as the MSP430 is much faster as the pic one. But it seems that Contiki itself is not quick enough to handle this, so we added a pause of a few milliseconds between every packet until no error occurs and every data has arrived correctly.