Universitatea POLITEHNICA din București
Facultatea de Automatică și Calculatoare
Departamentul Calculatoare

Lucrare de Disertație

Arhitectură pentru rețele wireless de senzori cu energy harvesting

Autor

Ioan Deaconu

Coordonator: Sl. Dr. Ing. Dan Ștefan Tudose

București, Iulie 2016

University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers
Computer Science and Engineering Department

Master's Thesis

Arhitecture for energy harvesting wireless sensor networks

Author

Ioan Deaconu

Supervisor: Sl. Dr. Ing. Dan Ștefan Tudose

Bucharest, July 2016

Contents

Abstract

Powering a device using solar generated energy can be difficult, especially when that device is meant to function constantly over a long period of time. In this thesis we will present an architecture for energy harvesting wireless sensor networks that can be used to develop solar powered applications. It will cover both hardware as well as the software needs in an energy harvesting wireless sensor network. The hardware will be composed of a new low power node designed to be a powerful development platform and an efficient energy harvesting module. The software is designed to efficiently use the stored energy by implementing a lightweight algorithm for scheduling data transmission.

Keywords Wireless Sensor Networks, Sparrow, Sparrow R, energy harvesting, scheduling, solar powered, dynamic scheduling

Acknowledgements

Foremost, I would like to express my gratitude to my advisor Dan Tudose for the continuous support that he provided.

Besides my advisor, I would also like to give special thanks to Dan Dragomir for providing corrections to this work.

My sincere thanks also goes to Adriana Drăghici, Alexandru Corneliu Olteanu and Andrei Voinescu for their insightful comments and moral support.

I thank my fellow labmates from the Robolab robotics group: Tudor Vișan, Andrei Mușat and Andrei Vasiliu for the sleepless nights we were working together before deadlines, and for all the fun we had in the past four years.

Last but not the least, I would like to thank my girlfriend and my family who have shown understanding and have given me the moral support needed.

Chapter 1

Introduction

A big problem encountered when developing an application for Wireless Sensor Networks is autonomy. Big batteries can be used to power the nodes, but because some can be deployed in locations difficult to reach, a simple task of changing the batteries becomes an impossible one.

The solution to this problem is powering the devices from alternative sources of energy, a process called energy harvesting. In recent years, energy harvesting has become more and more used in the field of Wireless Sensor Networks. There are plenty of alternative energy sources, such as solar cells, vibration absorption generators, wind mills, thermoelectric generators and others, that can be used to power the nodes or charge their batteries in order to become autonomous.

While the energy source problem has a solution, another problem appears in the form of finding a method to store the generated energy. A battery could be used to store the generate energy, but unfortunately, current technology allows to charge one for a few hundreds of cycles. Considering that a cycle would be used per day, in maximum 2 years, the battery will lose most of its original capacity. An alternative to the battery is using a super capacitor, which has a lifetime of 10 years or 500.000 cycles, but are more expensive and store far less energy than regular rechargeable batteries.

In order to alleviate the problem of the small amount of stored energy, a scheduling algorithm can be used. The main goal of the algorithm is keep the functional between two sessions of generated energy without losing power and to send as much data as possible. This is achieved by dynamically varying the frequency with which the node performs various tasks or sends data.

In this thesis we will describe the architecture of an energy harvesting wireless sensor network (EHWSN). We will present a new node and development platform, the Sparrow R, designed for low power and the problems we encountered when creating a EHWSN application. As the final part of the architecture, we

developed an efficient but lightweight algorithm for efficiently using the stored energy.

Chapter 2

Related Work

Energy harvesting has been studied for a long time, and despite of the advantage of free power, there are some downsides. Solar panels are frequently used in EHWSNs because they can generate a lot of energy but it is not consistent and predicting it represents a challenge to overcome.

In this chapter we will present the current state of the art in solar powered EHWSN.

2.1 Prediction generated energy

Because the solar energy is not constant, in order to predict the generated energy, a history of past-days weather conditions or generated energy must be taken into account. The state-of-the-art algorithm for this is Weather-Conditioned Moving Average (WCMA) [15]. In order to predict the generated energy in the next hour, it needs to keep a history of generated energy for the past 6 days. The results of the algorithm are shown to be impressive, with an average error of 9.8% in 45 days of testing.

Unfortunately, the algorithm requires the measurement of the generated energy in order to have an exact history of generated solar energy. Because of this, the algorithm is not feasible in applications where the node needs to be very low power, due to the overhead of estimating the generated power. An energy measurement chip can consume a few mW which can be almost all the generated energy by a small solar panel, for example on that could generate 20mW, much smaller than the one used in the paper, which could generate more than 300mW. A simpler solution for hardware as well as software must be found for those applications.

2.2 Using stored energy

The other downside of solar powered is the fact that in the night, rain or foggy conditions the panel will generate a very small or next to nothing amount of energy. In order to keep the node alive, transfer speed scheduling algorithms have been developed.

The common approach to optimal packet scheduling is using a water-filling algorithm [18] [10], where the time is divided into slots and given a level of energy to be used in that slot. For better power optimizations, this approach is modified into backward water-filling, directional water-filling, generalized iterative water-filling [17] for offline (deterministic) scenarios. Real world applications are stochastic, so in order to simulate online scenarios, Gaussian noise is added. The algorithms that can be used in this case are constant water level policy, energy adaptive water-filling[14], time-energy adaptive water-filling[13]. Because these are complex algorithms, simpler ones have been attempted, such as fuzzy power management[8] , where a table with predetermined levels is used as the main policy. What this mean is that the above algorithms will work best in high power scenarios, where leakage currents are small enough to be considered inexistent. Furthermore, the results of all the algorithms were obtained using simulated data with programs such as GreenCastalia[9]. Because the algorithms were never tested on a real device can mean that real conditions, like capacitor leakage or temperature variations are not taken into consideration.

All the above algorithms need to measure how much energy the node is using when performing different tasks. This is not feasible in real conditions because precise current measurement can only be performed at a high sample rate, which in turn consumes a large amount of energy. Considering the presented problems, we developed a lightweight and efficient algorithm that needs to know only the voltage of the capacitor.

Chapter 3

Sparrow R

Most of the nodes are build using the AVR processor [12] [16], an 8-bit architecture. Few attempts have been made using the newer ARM Cortex-M3 32-bit architecture. The benefits of higher computing power are presented here [11], where a 4.6 throughput can be obtained compared to an 8-bit CPU over the same wireless network. Unfortunately, the downside is higher power consumption that can create difficulties for certain power supplies. Even though the same average power consumption can be obtained, due to higher power peaks, the voltage of the power supply can drop to a lower level than the minimum required voltage of the node.

The current nodes, Sparrow V3.2 and SparrowV4, use an AVR 8-bit architecture and run at a speed of 16MHz. For our solar powered node, we wanted to use a newer, more powerful and lower power microcontroller. Based on the previous requirements, we have chosen an ARM Cortex-M0+ 32-bit Atmel SamR21.

3.1 Performance

We present bellow the main differences between the two MCUs, SamR21 and ATmega128RFA1 [2].

Being a 32-bit architecture, even though SamR21 consumes 5.5mA compared to 4.1mA of the Atmega128RFA1, for simple 32-bit integer addition, the SamR21 consumes only 49nJ per iteration while the 8-bit microcontroller consumes 274nJ (almost 5 times more). Considering performance figures, the SamR21 was 9 times faster with 403950 iterations per second while Atmega128RFA1 managed only 44890 iterations.

Testing the performance of the branch predictor, revealed that the M0+ is only 12% better than the older 8-bit counterpart when running at the same speed, but thanks to the frequency difference, it ends up being 3.36 times faster.

Criteria	Atmega128RFA1	SamR21
CPU Speed	16 MHz	48 MHz
CPU architecture	AVR 8bit	Cortex M0+ 32bit
CPU Power	4.1 mA	6.5 mA
Flash	128 kB	256 kB
RAM	16 kB	32 kB
Flash Endurance	50000	150000
Rx Consumption	12.5 mA	11.8 mA
Tx Consumption	14.5 mA @ 3.5mA	13.8 mA @ 4 dBm
Receiver sensitivity	-100 dBm	-101 dBm
Tx Max Power	3.5 dBm	4 dBm
Package	QFN64	QFN48 or QFN32

Table 3.1: Comparison between Atmega128RFA1 and SamR21

Criteria	Atmega128RFA1	SamR21	Total Advantage	Advantage per MHz
Integer Iterations	44890	403950	8.99	2.99
Branch Iterations	27782	93552	3.36	1.12
While(1) Iterations	191536	6693086	34.94	11.64

Table 3.2: Speed comparison

Criteria	Atmega128RFA1	SamR21
Integer Iteration	274 nJ	49 nJ
Branch Iteration	442 nJ	208 nJ
While(1) Iteration	64 nJ	2.9 nJ

Table 3.3: Energy efficiency comparison

The SamR21 microcontroller is almost the same as SamD21 [4], which is used in the Arduino Zero boards. This allowed us to use exiting code, but unfortunately a not well written one.

Even thought the Arduino software is well designed, it was not designed with low power consumption in mind. We will describe some of the problems encountered in the current software stack.

The first problem we noticed was that the Arduino Zero board had no sleep functionality implemented. The ideal idle current consumption should have been less than 5μA, tested and measured using a project created in Atmel Studio 7.0. The current consumption of the board was around 350μA. Further tests revealed that the USB device was always initialized, which accounted for the extra 200μA. The remaining of 150μA came from a default initializations of the pins as input pins, but this only lowered the current consumption to about 60μA. We kept searching for a cause, and discovered that the clock generators are never disabled at start-up, which accounted for about 30μA.

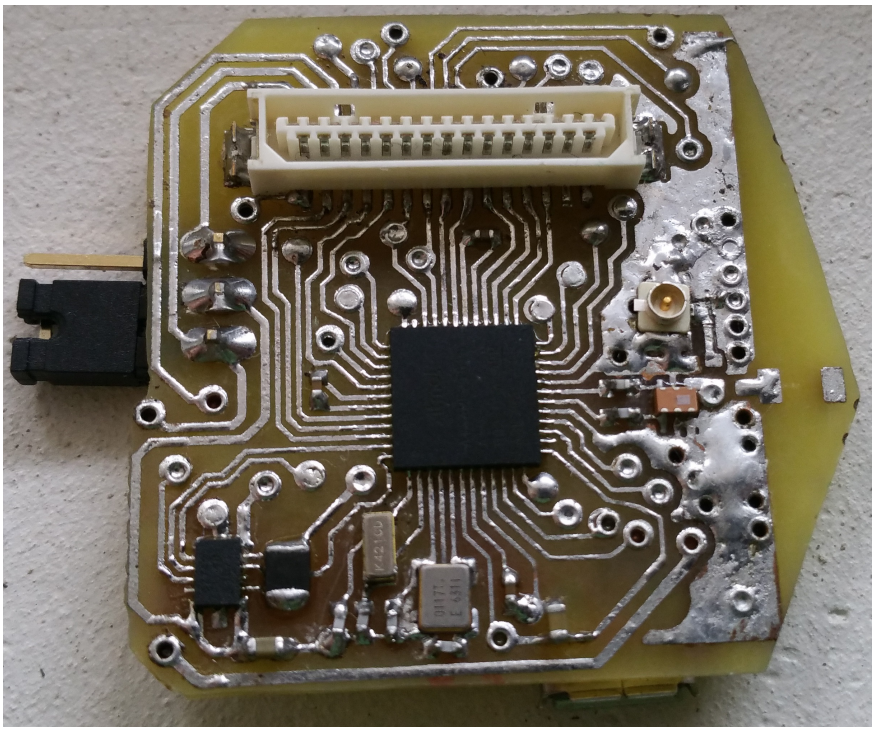

Figure 3.1: Sparrow R node

So far we managed to a decrease the idle current consumption for the platform from 350μA to about 30μA @ 3.2V, but it is still far from ideal. Surprisingly, lowering the voltage from 3.2V to 1.8V lead to decrease in sleep current consumption down to 3.3μA. When examining the power trace using a digital oscilloscope, we found that a very low frequency clock remains active, which at 3.2V has high spikes in power consumption.

Though we did not reach the goal of 5μA, we still managed a respectable 30μA @ 3.2V and less than 4μA @1.8V. Due to time constrains and the need to use the nodes in order to implement and test new features, we decided that for now this is acceptable, and for future revisions, we will come back and find the extra clock source.

Even the run current consumption was not ideal, instead of achieving the promised 70μA/MHz @ 3.2V, or around 3.5mA @ 48MHz, the microcontroller consumed 8mA @ 48MHz. We managed to reduce the current consumption to 5.5mA @ 48MHz, due to clock optimizations presented bellow.

The first modification was to change the clock of the peripheral interfaces, instead of 48 MHz, we run them at 12 MHz. Also if peripherals are not used, we completely disable them. Due to this, we ran into problems related to SERCOM implementation, a generic module that handles USART, SPI and I2C. It was working on Arduino Zero, because the CPU and the BUS were configured to

run at the same speed, but because of previous clock source modifications, the SERCOM did not set the correct speed. Also, there are 6 SERCOMs, and instead of enabling the clock for each one only when it is used, all of them were enabled, which lead to extra power consumption during run time.

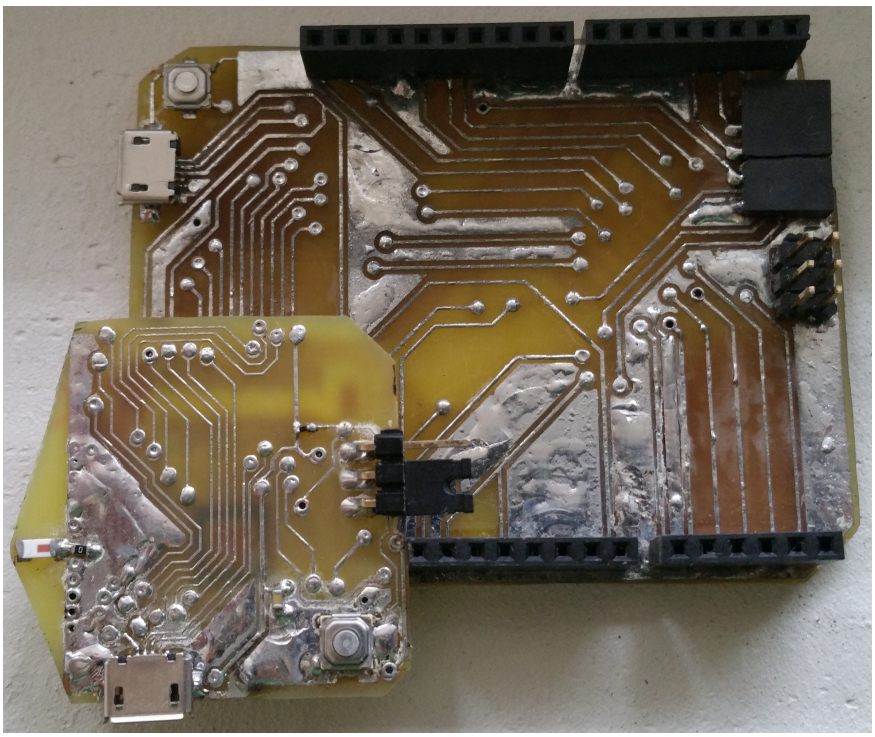

Figure 3.2: Sparrow R node mounted on Aduino compatible base

3.2 Hardware

Because not all sensors are designed to run at 1.8V up to 3.2V, we need to be able to dynamically change the operating voltage of the node. We used the TPS62742, a step-down switching DC/DC converter with up to 90% efficiency, voltage selectable output from 1.8V to 3.2V in 200mV steps, 360nA quiescent current and a special MCU controllable load output, with push-pull transistors. In inactive state the load line is pulled to GND and when active is pulled to VCC. When active it consumes 12μA, but compared to the controlled sensors, this should not be noticeable. Because this allows to completely power down the sensors, the "stand-by" current consumption is 0μA and it also eliminates the previous design problem of floating GND which allowed the sensors to "steal" power from other pins and not be properly disabled.

The node can be connected to an extension daughter board which fully respects the Arduino pinout. The advantage of this approach is that it allows to

easily test and prototype new configurations in order to prepare the project in the shortest time possible. Also existing hardware designed for Arduino can work with this board, which increases the number of compatible hardware. In total, 20 I/O pins are available, pins that can be used for connecting sensors, either on the daughter board, or directly on a specially designed board.

A jumper can select whether the Node is powered from USB or from other 2.1V+ voltage supplies. Through the same jumper a power measuring device can be used to monitor the total power consumption. In case the DC/DC converter is not needed the node can use other power sources.

3.3 Software

We implemented new modules designed for low power like sleep and power management, which can dynamical change the running voltage between 1.8V to 3.2V when requested, enable or disable the LOAD power line. This allows the user to select which voltage is better required for applications. For example, some sensors must be powered at exactly 2.8V while others at 2.5V or lower. Using this module, the user can select the desired voltage, use the sensors and then switch to the lowest voltage in order to obtain the best power consumption possible.

The RF module is an AT86RF233, integrated into the microcontroller. We desired to create an easy to use software stack, that will let the user focus on what to do with the platform and not how to do it. We integrated the module for the RF in the core of the platform, module based on this Arduino library [3]. Furthermore, we added extra features and fixed the existing bugs of the library. For example, in case the RF is constantly running in receive mode for more than 5 minutes, it is recommended to do Fine Tuning of the PLL clock in order to eliminate possible clock skews. Feature wise, when the module automatically receives a packet, it saves it locally together with the RSSI and LQI, which can be later read and used buy the user. The internal buffer is designed for 8 packets of 127 bytes of data, which amounts to 1 KB of ram. The buffer is cyclic, so in case the buffer is not read, the oldest data is discarded and replaced by a new one. This should not happen very often, because the buffer is large enough to handle all request, even for high bandwidth transfers.

If the user has a need to save the data in case of power failure, the microcontroller has an EEPROM like functionality which allows a 16KB region of flash to emulate EEPROM write endurance. The flash contains pages of 64 bytes, and the EEPROM has an overhead of 4 bytes, which leaves 60 bytes for actual data. Also, for each page, another page must be reserved for further use. The results is that out of 16KB used, the total amount of usable space left is $\frac{16*1024}{2} * \frac{60}{64} = 7680 bytes$. This should be more than enough for normal use because the normal endurance of 25k cycles of flash write and erase are increased to at least 150k, with typical

values reaching 600k cycles. When new software is uploaded, the EEPROM zone is completely erased.

For timekeeping when sleeping, the RTC functionality was implemented. Besides keeping the time, RTC provides alarm interrupts for a special date, which can be configured to be triggered every minute, every hour, every day, every month, every year, or only once. Together with another peripheral named EventSys, periodic interrupts are provided and the interrupt interval can range from once every second up to 128 times per second, with increments of power base 2.

Because the software and hardware are never perfect, a watchdog functionality is also implemented, in order to avoid code lock-up or hardware failure due to extreme environment conditions.

The software can be installed as a new board for Arduino 1.6.8, the latest iteration to date (May 2016). It was tested using Windows 8.1, but it should be fully compatible with other operating systems as well.

The node has native USB which allows for code upload and also serial interface over CDC. Because no extra components are needed, the same node can be configured to act as a gateway or as a leaf.

3.4 Power Consumption

Having a very low power consumption, a small solar panel together with a small capacitor can be used as the main power supply.

For example, when a 1F capacitor is used with the voltage ranging from 3.3V to 1.9V the equivalent battery capacity would be

$$\frac{F * (Vi - Vf)}{t} = \frac{1F * (3.3V - 1.9V)}{3600s} = 0.388mAh$$

The node is equipped with a DC/DC converter that can bring substantial power savings, depending on the voltage of the power supply. A LDO wastes a lot off energy as the delta between the voltages increases, were as DC/DC works best when the delta increases, or putting it simple, if an LDO outputs 1.8V and a chip consumes 5mA, the input current is almost the same 5mA regardless of the input voltage. So even though the chip consumes 9mW, the total power consumed is actually 25mW. If in the same situation, a DC/DC with a 90% efficiency is used, then the input current would be :

$$Iin = \frac{Iout * Vout}{Vin * Efficiency} = \frac{5mA * 1.8V * 100}{5V * 90} = 2mA$$

We can ignore the quiescent current because it is very small, around 360nA. The total power consumption in this case is 10mW for an output power of 9mW

compared with the LDO which consumes 25mW in order to output just 9mW, a 250% difference between them.

In a real world situation, when a battery or a capacitor is used, the difference between them is smaller. We will present two cases, in order to better understand the influence of higher voltage supply.

The minimum voltage will be 1.9V for DC/DC as well as for LDO. The current consumption of the chip is 5.5mA, and we assume a capacitor of 1F.

Case 1: Capacitor charged to 3.3V.

$$T_{LDO} = \frac{C * (V_i - V_f)}{I} = \frac{1F * (3.3V - 1.9V)}{5.5mA} = 254.54s$$

$$E_i = \frac{C * V_i^2}{2} = \frac{1F * (3.3V)^2}{2} = 5.445J$$

$$E_f = \frac{C * V_f^2}{2} = \frac{1F * (1.9V)^2}{2} = 1.805J$$

$$E = E_i - E_f = 3.64J$$

$$P = V * I * E_{ffiiciency} = \frac{1.8V * 5.5mA * 90}{100} = 11mW$$

$$T_{DC} = \frac{E}{P} = \frac{3.64J}{11mW} = 330.9s$$

$$Advantage = \frac{T_{DC} - T_{LDO}}{T_{LDO}} = \frac{330.9s - 254.54s}{254.54s} = 30\%$$

Case 2: Capacitor charged to 5V.

$$T_{LDO} = \frac{C * (V_i - V_f)}{I} = \frac{1F * (5V - 1.9V)}{5.5mA} = 563.63s$$

$$E_i = \frac{C * V_i^2}{2} = \frac{1F * (5V)^2}{2} = 12.5J$$

$$E = E_i - E_f = 10.695J$$

$$T_{DC} = \frac{E}{P} = \frac{10.695J}{11mW} = 972.27s$$

$$Advantage = \frac{T_{DC} - T_{LDO}}{T_{LDO}} = \frac{972.27s - 563.36s}{563.36s} = 72.5\%$$

At 3.3V the advantage is not that big, but at 5V we nearly double the battery life. Furthermore, using a LIPO battery will increase the advantage of the DC/DC.

Chapter 4

Energy Harvesting

This chapter presents the energy harvesting platform.

It is difficult to chose the best solution for solar energy harvesting, due to power requirements of different applications. In this chapter we will present the hardware that we used to generate and store solar energy.

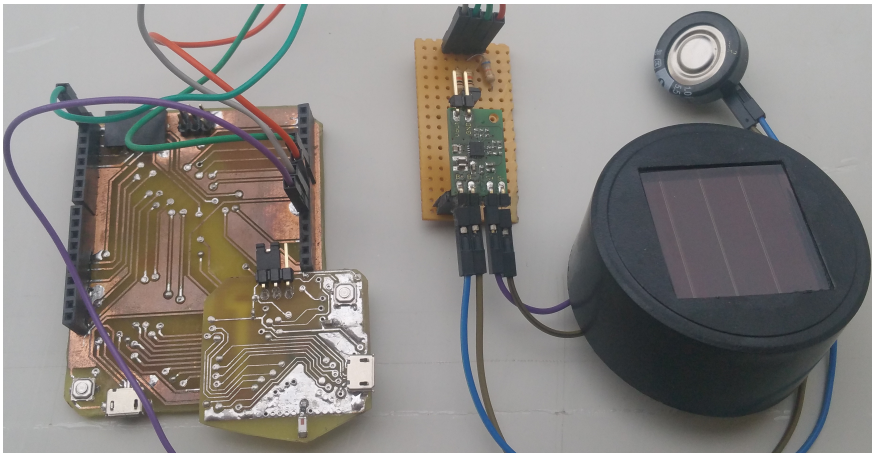

Figure 4.1: Setup for solar energy harvesting

4.1 TI BQ25504

The main component of an energy harvester is the boost converter needed to raise the voltage of the solar cell in order to be able to charge a battery or a super-capacitor. The chip selected by us is the TI BQ25504, a very efficient boost converter with battery management for energy harvesting.

The chip can begin charging from a low voltage of only 330mV, has a very small quiescent current, less then 360nA and once started, it can harvest energy until the voltage drops down to 80mV.

Another advantage is that the output voltage can be selected from 2.5V up to 5.25V, which can be used, as demonstrated in previous chapter, to charge a capacitor to a higher voltage in order to dramatically increase the total amount of stored energy.

Four our experiment, we have selected an output voltage of 3300mV.

4.2 Super-capacitor

The advantages of the super-capacitor over the rechargeable battery are as follows:

- it can charge and discharge almost instantaneously
- it has a very high number of charge/discharge cycles
- it does not suffer from the same aging symptoms as a battery
- it more eco-friendly than a standard battery
- it will allow a longer maintenance-free time than a battery

The disadvantages of the super-capacitor are :

- it can store a much smaller amount of energy than similar sized batteries
- it is very expensive
- it operates at low voltages and may require a charge pump to raise the voltage
- higher current leakage.

The main problem of a capacitor is current leakage. The bigger the capacitor, the bigger the current leakage. Also the technology with which they are built, the number of cycles or age can also have a profound effect on leakage current.

Even though it does not seem much, if a 1F capacitor has a self discharge rate of 6uA [6], this is more than the sleep power of the Sparrow R. Even more, if using a 25F capacitor the leakage current is increased to 45uA, which can be equivalent to the same power consumption of the node sending data every few seconds. Because of this, the total energy that can be stored is greatly affected by the duration needed for the node to use the super-capacitor as a power supply.

Charging is also a problem. If a panel that can supply 50uA, for a 1F capacitor is more than enough to charge the capacitor and supply power to the node, for a

25F capacitor that power is barely enough to maintain the stored energy, without even powering the node.

For our experiments we have chosen two 1F capacitors rated at 5V.

4.3 Solar Panel

We can use any solar panel, as long as it is rated up to 3V and has enough power to charge the selected capacitor.

Because we do not need a large amount of energy, a small solar cell of just 60mW is more than enough to fully charge the capacitor in just 5 minutes of sunny weather. This means that regardless of the meteorological condition, we would be able to charge the capacitor daily.

Chapter 5

Software Implementation

In this chapter we will present a simple yet efficient scheduling algorithm for single hop networks.

State-of-the-art algorithms require to measure the consumed energy in other to dynamically schedule transmission task. This is not very desirable in real world, where a lot of variables can generate a big error in the estimated energy left

- Large changes in temperature can alter the total energy stored
- Leakage currents, that can vary between charges and different levels of stored energy
- Differences between supposed identical super-capacitors.

In order to simply the algorithms for the user to deploy them faster in a real EHWSN applications, we have implemented and tested a simple dynamic scheduling algorithm for transfered data.

The algorithm can be considered to have very basic water-filling policy, with one slot that is the current discharge period. This simplifies the implementation and reduces the computational requirements. It is designed to be run with an energy harvesting module that has a super capacitor as energy storage unit. Because of this, we simplified the predictor in order to obtain O(1) complexity.

The capacitor does not discharge very linearly, but in the tests we have conducted, the error is small enough to allow us to estimate the remaining time without the requirement of measuring how much energy did the capacitor stored and how much energy was consumed. Figure 5.1 is an example of how the capacitor will discharge.

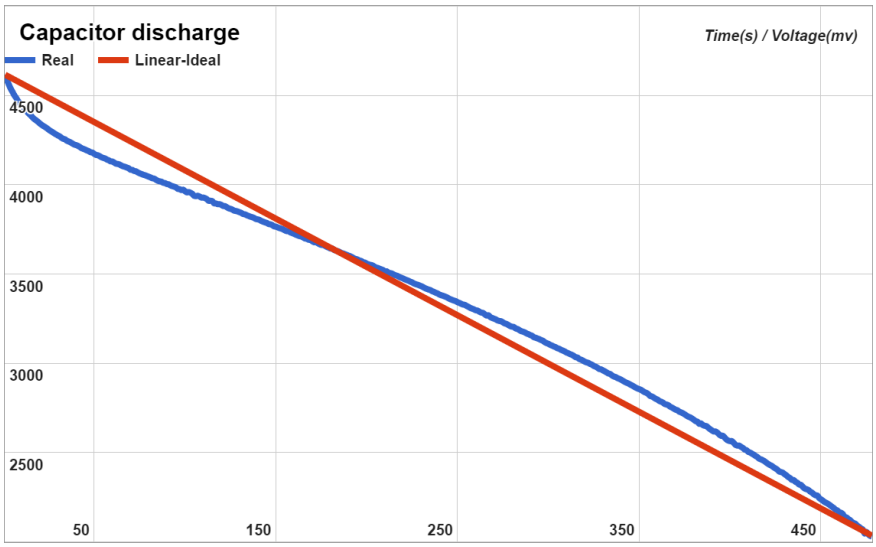

Figure 5.1: 1F Electrolytic Double Layer capacitor discharge

The algorithm has a total time that it needs to respect, while trying to maximize the total number of packets sent. In order to achieve this two simple requirements, it dynamically estimates the remaining time and, according to the difference between the estimation and the remaining time, it keeps the same send frequency or alters it. Because when lowering the speed, the total consumed energy will not be halved, due to leakage and sleep current, a penalty is imposed, one that we selected at 75%. Furtermore, in order to make sure that the deadline is respected, under a certain voltage level the speed is halved and kept until the energy is depleted or the capacitor is recharged. Every time the frequency changes, the interval used to compute the estimated time is updated.

The algorithm is very versatile, with many customization options, depending on the type of hardware and requirements of the application in which it is used.

What can be changed and it is recommended to be changed is presented in the list bellow. Other customization options exists, but it is not recommended to changed them.

- DEFAULT_TARGET_TIME The default deadline for the algorithm. After one iteration it is dynamically adjusted.
- TIME_PERCENTAGE_PENALTY Alters the estimated time by either reducing it or increasing it.
- TIME_EXTRA_REMAINING When computing the new frequency, this is added to the remaining time in order to allow for longer running period.
- SPEED_DECREASE_PENALTY When the transmission speed is decreased, a penalty must be applied.

Algorithm 1 Send frequency scheduling Algorithm

```
1: procedure ALGORITHM(voltage, time)                                   ▷
2:     determine current state Charging/Discharging
3:     if changed from Charging to Discharging then
4:         Calculate the new target time
5:         Calculate the frequency based on total number of sent data
6:         changedVoltage ← voltage
7:     end if
8:     if current state is Discharging then
9:         if voltage < MIN_COMPUTE_DELTA then
10:            sendFreq ← sendFreq/2
11:        end if
12:        deltaVoltage ← changedVoltage − voltage
13:        if deltaVoltage > MIN_VOLTAGE_DELTA then
14:            Estimate the new remaining time
15:            Calculate remaining time until deadline
16:            newSendFreq ← sendFreq ∗ estimated ∗
   percentagePenalty/remaining
17:            if newSendFreq! = sendFreq then
18:                Change the speed
19:                changedVoltage ← voltage
20:            end if
21:        end if
22:    end if
23: end procedure
```

- MIN_SEND_FREQ Minimum packets per hour for the algorithm. Minimum value that can be set is one per hour
- MAX_SEND_FREQ Maximum packets per hour for the algorithm. Maximum value that can be set is one per second, or 3600 per hour.
- DEFAULT_SEND_FREQ The default packets per hour before the algorithm stabilizes.
- MIN_VOLTAGE_COMPUTE The minimum voltage under which the algorithm will stop adjusting the frequency
- MIN_VOLTAGE The minimum voltage considered by the algorithm when computing the estimated time and the new send frequency.
- MIN_VOLTAGE_DELTA The delta voltage over which the algorithm will start to compute the estimated time and the new send frequency.

The algorithm will recalculate the new deadline every time, so in case the sun will rise sooner or later the target time will be automatically adjusted. In order

to compensate for the situation in which the sun will rise later, the algorithm always tries to keep the node alive for a longer time than the given deadline. The extra time the node can be kept alive can vary according to the duration, from 10 minutes for a 4 hours deadline, to 1 hour for a 12 hour deadline.

Chapter 6

Evaluation

Our goal was to deliver a solution that could be deployed in an application as fast as possible, so instead of running software simulations, we decided to implement and test the algorithm on the new node Sparrow R. The algorithm is implemented in C, and compiled with gcc-arm-none-eabi-4.8.3-2014q1 on Arduino 1.6.4.

We selected 2 super-capacitors of 1F and 5V maximum rating, one using Electrolytic Double Layer (EDLC) technology and other using Aerogel technology. We fully charged the capacitor and then let the algorithm decide how fast the data should be transmitted in order to reach the time deadline.

The only input needed is the current voltage of the capacitor, read using a voltage divider that is controlled by an n-mos transistor in order to reduce the power dissipated by the divider. The node will run a task that simulates sensors readings and other processing through a delay of 100 ms. The network is configure as single-hop, and the data sent through RF has the length of 45 bytes.

When beginning with a deadline of 4 hours, in the first run, the node manages to execute 1593 task and be functional for a total time of 4 hours and 18 minutes. The second run, with a new deadline of 4 hours and 11 minutes, the node ran 1631 tasks for a total time of 4 hours and 35 minutes. The results of the test are present in figure 6.1, were we can see that both runs have a similar frequency curve, and even discharge voltage.

With a longer, more realistic deadline of 8 hours, the node managed to transmit 785 times for a duration of 8 hours and 14 minutes.

$$E = E_i - E_f = 3.64J$$

$$P_{4h} = \frac{E}{T_{4h}} = \frac{3.64J}{258 * 60s} = 235\mu W$$

$$P_{8h} = \frac{E}{T_{8h}} = \frac{3.64J}{494 * 60s} = 122\mu W$$

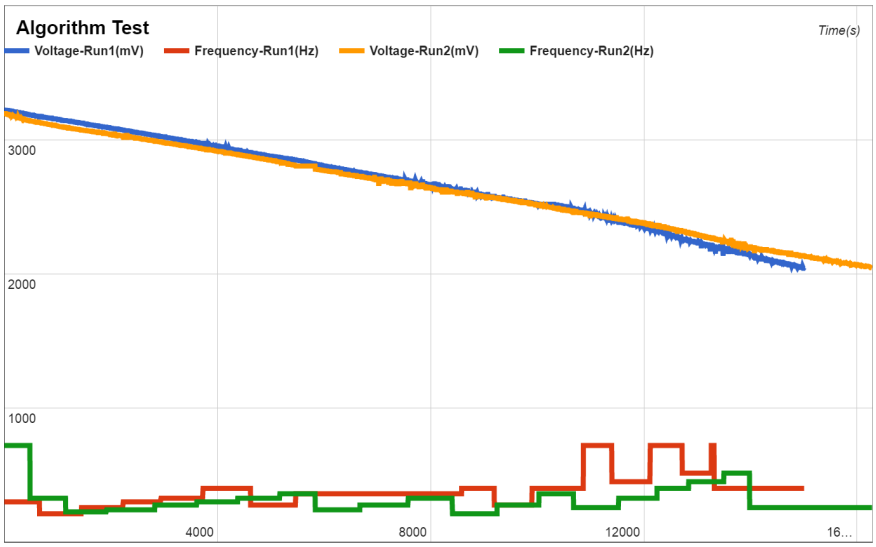

Figure 6.1: Two runs of the algorithm with EDLC capacitor and 4 hours deadline

When the total running time is doubled, the average power consumption is halved, proving that the algorithm is working. Unfortunately, the total number of send data is twice as small, which indicates that the combined power used by the node when sleeping and the leakage of the capacitor is starting to influence the total number of sent data. This is best shown by the difference between the energy used to send 1 frame in the 4 hours situation compared to 8 hours.

$$E_{frame4h} = \frac{E}{Packets_{4h}} = \frac{3.64J}{1631} = 2.23mJ$$

$$E_{frame8h} = \frac{E}{Packets_{8h}} = \frac{3.64J}{785} = 4.64mJ$$

We ran into problems when a longer deadline of 12 hours was tested, mainly because the self discharge of the capacitor combined with the idle current consumption of the node is high enough to waste more than 75% of the energy. This had a significant impact on the total number of executed tasks. The first run of the test send data 143 times and lasted for 13 hours and 10 minutes. The second run started with a more realistic speed of 10 task per hours instead of 600, but the total number of sent data dropped to 18 with a total duration of 13 hours and 15 minutes.

The 12 hours test revealed that a 1F capacitor is barely enough for 12 hours with our demo test and that a bigger capacitor might be needed. Because we had two types of capacitors, shown in figure 6.2, we were curios to see if there is a difference between them. What we found was a bit of surprise, reveled in figure 6.3 , especially the lowest voltage at which the node stopped working. The Aerogel capacitor had a lower voltage of 1950mV compared to the ELDC

of 2050mV. However, the EDLC capacitor managed to transmit more data than the Aerogel, even when the algorithm was tweaked to take into consideration the lower working voltage of the Aerogel capacitor. The EDLC managed to send data 1642 times, while the Aerogel capacitor managed only 1254 data transfers.

Figure 6.2: 1F different capacitors, Left EDLC - Right Aerogel

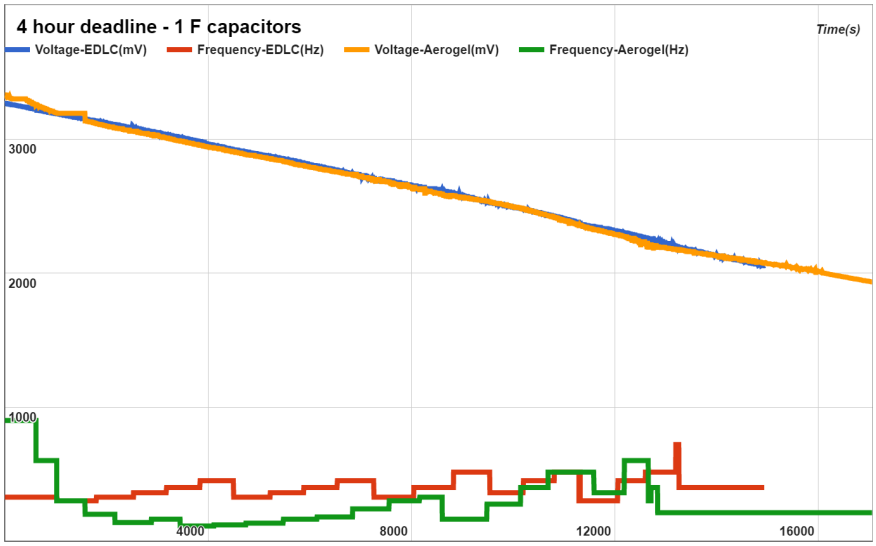

Figure 6.3: 1F different capacitors test, Aerogel vs EDLC

Chapter 7

Conclusions

The main goal of this thesis was to present the complete architecture of a Energy Harvesting Wireless Sensor Network. We have presented a brand new node designed to be a new development platform that can be used to bring new ideas to life. One of those ideas is dynamically varying the transmission speed in order to keep alive a node until the energy source (super-capacitor) can be recharged. We implemented an algorithm that we tested on the node, something that our research found that is rarely done, and proved that it is able to keep the node alive and efficiently use all the stored energy.

Because the algorithm only needs to read the voltage of the capacitor, the hardware requirements are very small and it can be easily deployed on existing hardware with little to no modification.

This research can be continued and for this we propose some directions presented in the next section.

7.1 Future Work

Even though the algorithm is functional, there is always room for improvement. We would like our work to be continued with an optimization of the algorithm for better time estimation. Taking into consideration past weather in order to predict when we can start to generate power could help us to better determine the new target time, so that in case of very cloudy weather, we could adjust the deadline and hopefully, keep the node alive until the sun will be able to recharge the super-capacitor.

The evaluation of the algorithm revealed at least two more areas in which improvements are needed. The first is revealed by the test of the Aerogel capacitor, in which the send frequency increased close to the deadline. This is not a negative

aspect, as the deadline was reached, but a smaller difference between consecutive changes in frequency might allow for a more constant energy discharge.

The second one is revealed by the 12 hours discharge test. The algorithm lowered the send frequency down to 1 data per hour, but the time interval used to determine the new frequency reached 10 hours. A good research direction would be to check if the frequency has changed in the last hour. If not, then the algorithm will need to forget the previous voltage interval used for estimation of the remaining duration, in order to follow the discharge curve.

Another point not covered in this thesis is multi-hop networks. We tested the algorithm in a simple single-hop scenario, but a multi-hop test is needed to know if the algorithm needs improvements for this scenario.

Bibliography

[1] Arduino. https://www.arduino.cc. Accessed: 2016-06-05. [cited at p. -]

[2] Atmega128rfa1 datasheet. http://www.atmel.com/Images/doc8266.pdf. [cited at p. 7]

[3] Arduino-at86rf233. https://github.com/msolters/arduino-at86rf233. Accessed: 2016-06-05. [cited at p. 11]

[4] Atmel samd21 datasheet. http://www.atmel.com/images/atmel-42181-sam-d21_datasheet.pdf, . Accessed: 2016-06-05. [cited at p. 8]

[5] Atmel samr21 datasheet. www.atmel.com/Images/Atmel-42223SAM-R21_Datasheet.pdf, . Accessed: 2016-06-05. [cited at p. -]

[6] Ultra-capacitors. http://www.maxwell.com/images/documents/Ultracapacitors_Overview_Flyer_3000615-2EN.pdf. Accessed: 2016-06-20. [cited at p. 15]

[7] Musat Andrei-Alexandru, Tudose Dan, and Deaconu Ioan. Geodynamics monitorization using wireless sensor networks. *CSCS*, 2015. [cited at p. -]

[8] Faycal Ait Aoudia, Matthieu Gautier, and Olivier Berder. Fuzzy power management for energy harvesting wireless sensor nodes. In *IEEE International Conference on Communications (ICC16)*, 2016. [cited at p. 6]

[9] David Benedetti, Chiara Petrioli, and Dora Spenza. Greencastalia: an energy-harvesting-enabled framework for the castalia simulator. In *Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems*, page 7. ACM, 2013. [cited at p. 6]

[10] Longbo Huang and Michael J Neely. Utility optimal scheduling in energy-harvesting networks. *IEEE/ACM Transactions on Networking (TON)*, 21(4):1117–1130, 2013. [cited at p. 6]

[11] Raja Jurdak, Kevin Klues, Brano Kusy, Christian Richter, Koen Langendoen, and Michael Brünig. Opal: A multiradio platform for high throughput wireless sensor networks. *Embedded Systems Letters, IEEE*, 3(4):121–124, 2011. [cited at p. 7]

[12] Shri R. N. Shukla Mridula Maurya. Current wireless sensor nodes (motes): Performance metrics and constraints. *International Journal of Advanced Research in Electronics and Communication Engineering*, 2(1), 2013. [cited at p. 7]

[13] Omur Ozel, Kaya Tutuncuoglu, Jing Yang, Sennur Ulukus, and Aylin Yener. Transmission with energy harvesting nodes in fading wireless channels: Optimal policies. *Selected Areas in Communications, IEEE Journal on*, 29(8):1732–1743, 2011. [cited at p. 6]

[14] Omur Ozel, Jing Yang, and Sennur Ulukus. Optimal broadcast scheduling for an energy harvesting rechargeable transmitter with a finite capacity battery. *Wireless Communications, IEEE Transactions on*, 11(6):2193–2203, 2012. [cited at p. 6]

[15] Joaquin Recas Piorno, Carlo Bergonzini, David Atienza, and Tajana Simunic Rosing. Prediction and management in energy harvested wireless sensor nodes. In *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pages 6–10. IEEE, 2009. [cited at p. 5]

[16] Andrei Voinescu, Dan Tudose, and Dan Dragomir. A lightweight, versatile gateway platform for wireless sensor networks. In *Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition*, pages 1–4. IEEE, 2013. [cited at p. 7]

[17] Zhe Wang, Vaneet Aggarwal, and Xiaodong Wang. Iterative dynamic water-filling for fading multiple-access channels with energy harvesting. *Selected Areas in Communications, IEEE Journal on*, 33(3):382–395, 2015. [cited at p. 6]

[18] Jing Yang and Sennur Ulukus. Optimal packet scheduling in a multiple access channel with energy harvesting transmitters. *Communications and Networks, Journal of*, 14(2):140–150, 2012. [cited at p. 6]

List of Figures

Listings