Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2015

Diogo Micael Repas Curto

Sistemas de Deteção por Infravermelhos De Muito Baixo Consumo
Low-Power, Highly Reliable IR Range Detection Systems

Diogo Micael Repas Curto

Low-Power, Highly Reliable IR Range Detection Systems

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e de Telecomunicações, realizada sob a orientação científica do Professor Doutor Pedro Fonseca, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Rui Manuel Escadas Ramos Martins
Professor Auxiliar, Universidade de Aveiro

vogais / examiners committee

Professor Doutor Vítor Manuel Ferreira dos Santos
Professor Associado, Universidade de Aveiro (Arguente Principal)

Professor Doutor Pedro Nicolau Faria da Fonseca
Professor Auxiliar, Universidade de Aveiro (Orientador)

agradecimentos / acknowledgements

Por toda a ajuda e companheirismo ao longo destes últimos meses, agradeço à malta da Exatronic.

Quero agradecer o apoio incondicional da minha família pelo enorme esforço que fizeram para me suportarem ao longo de todos estes anos.

Aos amigos que sempre caminharam comigo lado a lado, nos bons e maus momentos, um eterno obrigado.

Por último, mas não menos importante, um grande obrigado à Soraia por conseguir tornar sempre tudo mais fácil.

Resumo

A eficiência energética é cada vez mais uma preocupação de engenheiros e da população em geral. Em sistemas alimentados a baterias, esta preocupação torna-se mais evidente quando as pessoas interagem com estes diariamente. É então frustante quando a uma bateria descarregada impossibilita a utilização destes sistemas.

Um caso particular de sistemas que muitas vezes são alimentados por baterias são as torneiras automáticas. Estes sistemas necessitam de constante manutenção, quer devido à descarga das baterias, quer devido a falhas na deteção de presença. O princípio de funcionamento destes sistemas baseia-se essencialmente numa deteção por infravermelhos com recurso a um pequeno circuito de ativação de uma electro-válvula.

Nesta dissertação foi proposta uma implementação semelhante com algumas alterações. Utilizaram-se técnicas de baixo consumo, algoritmos de deteção por infravermelhos e ainda recolha de energia para aumentar a duração da bateria. Ao usar um microcontrolador para executar as tarefas requeridas, foi adicionada ao sistema alguma inteligência. Foi ainda estudada a possibilidade de tornar o sistema completamente autónomo em termos de geração e consumo de energia.

Embora a auto-suficiência não tenha sido alcançada, foram obtidos resultados importantes que poderão contribuir para melhorar o desempenho dos sistemas deste género.

Abstract

Energy consumption is one of the major concerns amongst engineers and general population. In battery powered systems, when people interact with them in a daily basis, this concern is even more evident. It is frustrating when a depleted battery makes impossible its normal use.

A particular case of a battery powered system is the automatic faucet. These need constant maintenance to replace dead batteries and even due to failures in presence detection. The working principle of these systems is essentially based in an infrared detection followed by a activation circuit of an electro-valve.

In this dissertation a similar, with some changes, implementation was proposed. The use low-power techniques, infrared detection algorithms and energy harvesting to increase battery duration. By using a microcontroller to perform the required operations, some intelligence was given to the system. It was also verified the possibility to make the system self sustainable in therms of energy consumption and harvesting.

Although self-sustainability was not achieved, several important results were obtained which can contribute to improve the performance of similar systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Energy consumption is one of the major concerns amongst engineers and general population. The world is facing a phenomenal growth of demand for energy, in major part due to population growth, economic expansion and urban development. These factors increase the demand for more personal-mobility items, appliances, devices and services.

Engineers play a major role in energy consumption, since they can optimize it at the chip, board, box, system and network levels. At each of these levels, there are major gains that can be achieved. Several important results in low-power design are being used to limit energy consumption in all system components.

On the other hand, energy harvesting has been used to power wireless sensor networks and other battery powered small electronics. The goal is to extract intermittently available energy from surrounding sources to power or increase battery life of specific systems.

Recent advances in ultra-low-power microcontrollers have produced devices that offer unprecedented levels of integration for the amount of power they require to operate. These are systems with power saving schemes, such as shutting down when idle or waiting for some events. In fact, so little power is needed to run these devices that many sensors are going wireless and run from batteries and/or energy harvesting for years.

Unfortunately, batteries must be regularly replaced, which is a costly maintenance process. A more effective wireless power solution may be to harvest ambient mechanical, thermal, or electro-magnetic energy in the sensor's local environment.

1.1 Motivation

One of the most commonly used sensors are infrared detectors. These detection systems have a wide range of applications, going from alarm systems to automatic faucets or hand dryers.

From the engineering standpoint, these systems must be easily installed, highly reliable and often battery powered with high autonomy. Easy installation means a system that a technician removes from the box and installs it without complicated configurations or adaptations. High reliability means that it should not generate false negatives/positives in presence of ambient changes. Finally, high autonomy is necessary to reduce maintenance costs.

This dissertation intends to study methods to maximize power efficiency in a microcontroller-based "smart" faucet, starting from the hardware up to the software layer. It was also an objective to implement a highly reliable infrared detection system while consuming the

least amount of power possible.

In order to increase battery autonomy, an harvesting method is also proposed. This brought to surface an attempt to make the system self powered. In order to estimate the system's overall energy consumption and generation (using the harvester) a mathematical model was developed. This allowed to estimate not only battery duration, but also improve the system's overall efficiency by managing several variables. Last but not the least, a system's management proposal is developed, based on said model.

1.2 The Company

Exatronic - *Innovation Insight*, is a company specialized in research, development and industrialization of innovative solutions in the areas of information technologies, communications and electronics.

Since 1995, by integrating a complete offer, such as electronics, firmware, mechanics, technical support and dedicated logistics, Exatronic's mission is to present innovative solutions with integrated electronics for their client's businesses and/or products.

Having as starting point a simple idea or concept, Exatronic carries out the technical viability analysis, builds the whole research and development process, and provides the manufacturing and delivery of the solution to the end consumer.

1.3 Thesis Structure

Starting from the main characteristics of low-power infrared detection, chapter 2 gives insights on major issues when designing low-power systems. A brief introduction to hardware and software co-design is also presented. This chapter ends with a case study where several design techniques were applied to a specific application in order to maximize its efficiency.

Following the low power concept, a brief explanation of the architecture for energy harvesting and available sources is presented in chapter 3.

Infrared detection is a major topic that fills many books on its own. In chapter 4, the main concepts of infrared detection are presented. Several sensor types and its limitations are presented and in particular sensors from SHARP and VISHAY are introduced and tested. This chapter ends with a case study for a thermopile sensor from HAMAMATSU to materialize some previously introduced concepts.

Chapter 5 is dedicated to a specific application designed to use three major concepts presented in the previous chapters: the low-power design, energy harvesting and infrared detection. This chapter starts by describing the system to be implemented and a theoretical energy balance is performed. After this, the experimental results in terms of each major topic are assessed and a final conclusion is drawn.

Finally, chapter 6 presents the major conclusions of this thesis, starting by presenting the issues addressed by this work and the ones left open.

Chapter 2

Low-power systems design techniques

Embedded systems usually have in its core a microcontroller. The "brain" of the application, in order to be considered low power, needs to meet certain requirements. Besides being specifically designed to consume as little power as possible, it must provide tools to manage adverse situations with energy constraints, such as voltage drops in battery applications, low peak current, minimum acceptable battery duration and so on.

When defining a system as being low-power, it is usually defined by the application itself. Each one has its own parameters for what it needs to be low-power. In most cases, a low-power application is defined by its efficiency. The system needs to get the most work done in the least time as possible, with the minimum use of CPU while working as slow as possible and consume only the strictly necessary power [1].

With these thoughts in mind, is clear that low-power systems require optimization at all levels, from the microcontroller and hardware architecture up through the application layer.

This chapter will introduce the main power categories and design techniques for microcontroller based systems and applications to be as efficient as possible.

2.1 Power Categories

There are two major factors involved in the application design: dynamic power consumption when it is running and static power for when it is asleep [2].

Dynamic power, as expressed in (2.1), is affected mainly by the supply voltage and the charging and discharging of capacitances at the clock frequency. The parameter α is a scaling factor that varies when considering an entire MCU, f is the clock frequency of the CPU, C is the internal capacitance and V is the supply voltage.

$$P_{dyn} = \alpha \times f \times C \times V^2 \tag{2.1}$$

This is the power consumed when the CPU is running and processing data. From a systems designer point of view, the changeable parameters during system design are the supply voltage and operating frequency. This change is not arbitrary because they are dependent on each other.

In a more global view, dynamic power is not only associated to the CPU itself, but also to the peripherals, both internal and external. This is important since most microcontrollers have the ability to completely shutdown internal peripherals in order to save power.

External peripherals can also be shutdown by using, for instance, I/O pins to power those devices, or even to control a transistor working as a switch.

Static power encompasses the power required to maintain proper system operation while code is not actively running and is highly dependable on voltage, temperature, leakage and bias currents for analog circuits. This is considered when the CPU is in standby (idle) and normally waiting for an event to occur. In battery applications, this is the state in which the CPU stays longer and therefore consumes the most significant battery power. Most applications need memory to work properly and therefore data retention power must be taken into account when discussing static power.

In terms of embedded systems, the most common practice is to discuss current consumption instead of power consumption, due to the fact that, normally, the voltage supply is fixed to a predetermined range. Therefore, over this text, devices' characterization will be expressed in current consumption.

2.2 Performance Factors

When designing a low-power system, the first step is normally to do a power budget. It is necessary to evaluate the needed or allowable power modes and estimate for how long the system will be running to accomplish the determined objectives. It is also important to consider the use of the peripherals: the time they will run in each different power mode and so on [3].

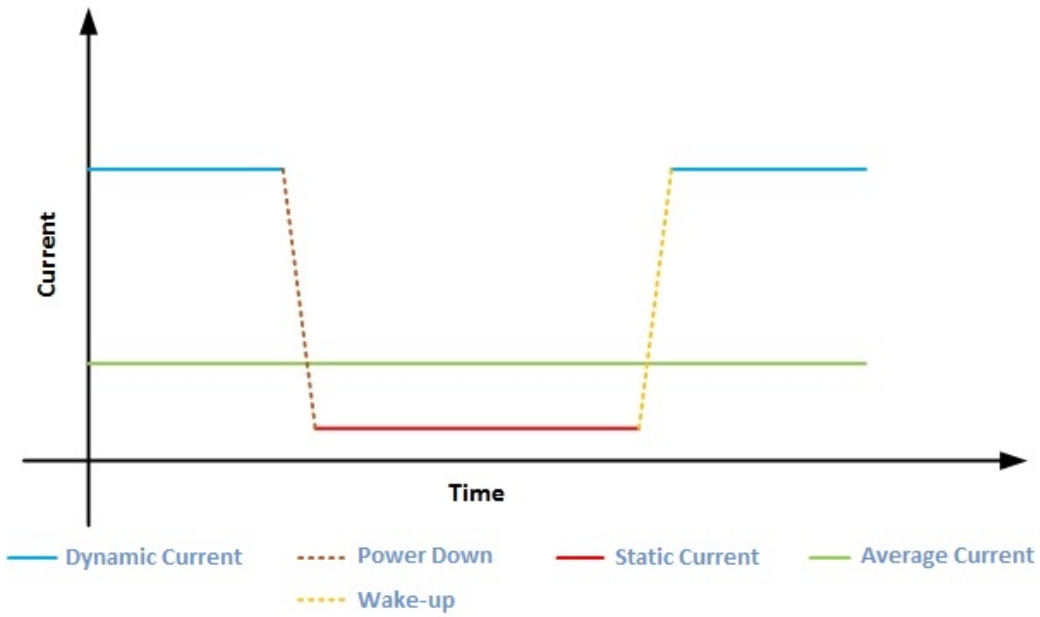


Figure 2.1: Generic system current consumption

After this, it is necessary to calculate the current consumption of each mode and the time spent on each mode to get an overall estimation of average current consumption for the

application as shown in figure 2.1 and expressed in (2.2), where I_{AVG} is the average current consumption, I_{dyn} and I_{st} are the dynamic and static current respectively, t_{dyn} and t_{st} are the respective times. This allows us to get an overall view in what is needed from the power-supply and the expected battery life. It also allows to get a good view of what parts of the system need more focus in order to minimize current consumption.

$$I_{AVG} = \frac{I_{dyn} \times t_{dyn} + I_{st} \times t_{st}}{t_{dyn} + t_{st}} \tag{2.2}$$

With these thoughts in mind, next sections will present microcontroller features capable of managing the current consumption of specific modules and tasks.

2.2.1 Supply Voltage and Clock Speed

Supply Voltage Range

When considering a reduction in the supply voltage in order to reduce the power consumption, there is a limitation that cannot be overlooked. The minimum supply voltage is limited by the operating frequency, that is, to reduce the supply voltage is also necessary to change the frequency. As an example in figure 2.2 is shown the frequency limitations depending on the supply voltage of a microcontroller from ATMEL [4]. The ATMEL AVR XMEGA is a family of low power, high performance, and a peripheral rich 8/16-bit microcontrollers based on the AVR enhanced RISC[1] architecture.

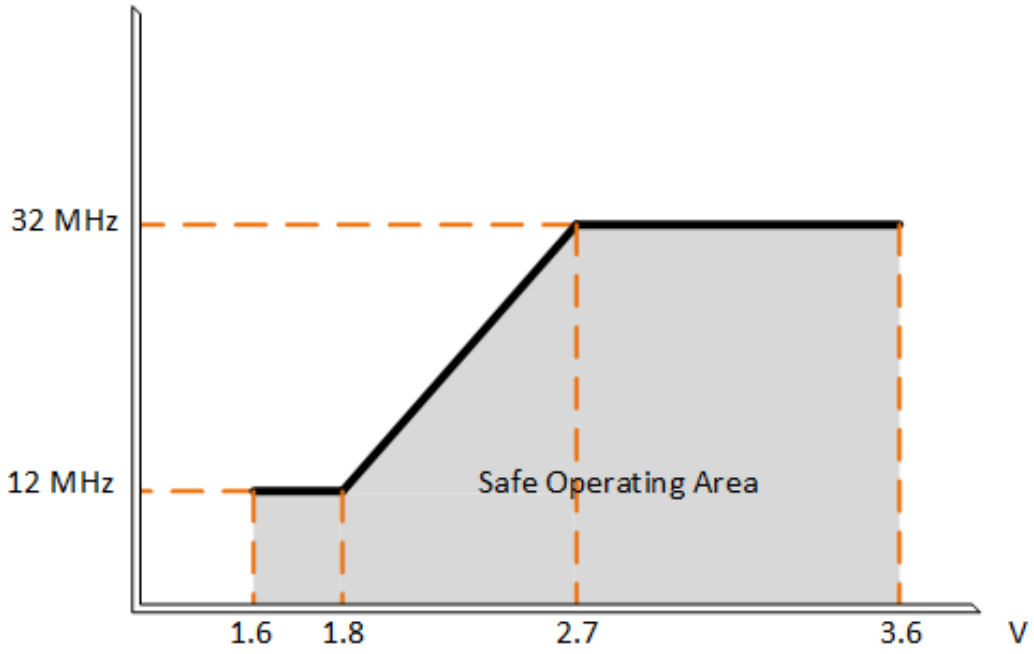


Figure 2.2: Frequency vs Supply voltage.

[1]Reduced Instruction Set Computer

Clock Speed

In an application that runs for a specific time and then goes back to sleep, is important to consider the trade-off between speed and current consumption, that is, evaluate what is best: execute at really low speeds or run fast and then sleep for a longer period.

In order to explain the statement above, an example is provided in figure 2.3 for active current consumption. It was obtained by taking a generic application with a certain amount of work that needs to run every ten seconds for five different clock speeds: 32 MHz, which in figure 2.3 is represented by IAVG32, 8 MHz by IAVG8, 2 MHz by IAVG2, 1 MHz by AVG1 and 31 kHz represented by IAVG31k.

As can be seen, executing at 31 kHz is always worst than higher speeds. This occurs due to the very low execution speed the microcontroller is almost always working instead of in a Deep-Sleep state [5] [6].

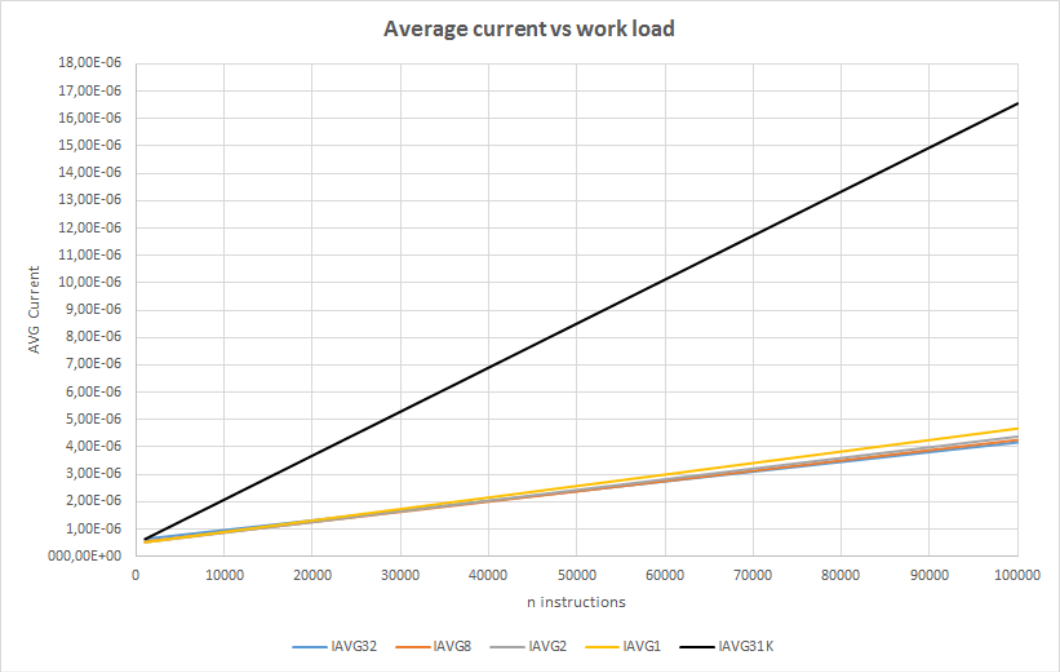


Figure 2.3: Clock speed average active current vs work load.

A better overview of this case is shown in figure 2.4. Here, the 31 kHz average current was removed. As the work load starts to increase, the most efficient frequencies are the highest. This indicates that executing at low speeds is only better if the work load is small.

These figures were obtained based on (2.2). More explicitly:

$$\begin{cases} i_{dyn}(f) & = & i_{DD}(f) + i_{\mathrm{WU}}(f) + i_{\mathrm{PD}}(f) \\ t_{dyn}(f) & = & t_{\mathrm{n\ inst}}(f) + t_{\mathrm{WU}} + t_{\mathrm{PD}}(f) \\ I_{st} & = & I_{\mathrm{DS}} \\ t_{st} & = & t_{\mathrm{DS}} \\ t_{dyn}(f) + t_{st} & = & 10 \text{ s} \end{cases}$$

Where $i_{dyn}(f)$, $i_{DD}(f)$, $i_{\mathrm{WU}}(f)$ and $i_{\mathrm{PD}}(f)$ are the dynamic, active, wake-up and power-down currents depending on the frequency. $t_{\mathrm{n\ inst}}(f)$ is the amount of time it takes to perform

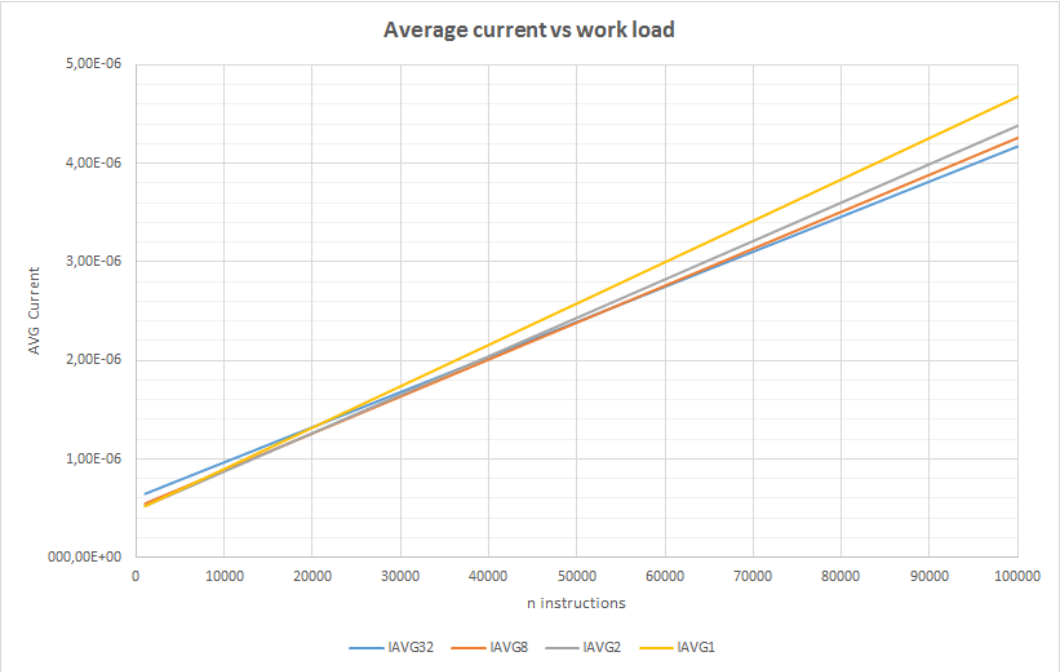


Figure 2.4: Clock speed average active current vs work load.

the instructions and finally, I_{DS} is the static current in the deep-sleep mode.

Another important aspect of frequency reduction to consider is the fact that in order to maintain the logic level, a certain amount of current must be used. This is necessary to maintain the charge level of the intrinsic capacitances of the microcontroller (recall (2.1)). For lower frequencies, this logic level must be held for longer, increasing the current consumption. When using higher frequencies, the voltage (logic) level decay is negligible making these more efficient than lower frequencies. An example, from Silicon Labs, is shown in figure 2.5 [7].

When considering running the application at higher speeds, there is another factor that rises as already shown in this section: most microcontrollers need higher supply voltages in order to run faster. Therefore, to avoid errors in code execution, another feature must be used: a low-voltage detector, that enables, for instance, to reduce clock speed in runtime and therefore extend battery life. There is another feature useful both at higher and lower speeds called the Brown-out Reset. This allows to protect the application from errors as batteries die, or when high peaks of dynamic current bring the supply voltage to an unsupported level, resetting the microcontroller in both cases. These circuits are particularly important if the system is battery powered. Since running at higher speeds requires more current and, as shown in figure 2.2, the clock speed may not be supported, is necessary to start slower and increase the working frequency while monitoring the voltage level, stopping near the MCU shutdown point. This allows the application to run as fast as possible and as long as the battery allows.

In some applications, the relevant mode is not when the application is running (active mode). This happens when the system spends a large portion of his time in a Sleep/Low-Power mode making this the most relevant mode for current consumption. Meaning that the most important mode to consider really depends on the duty cycle between the various sleep and active modes. [8]

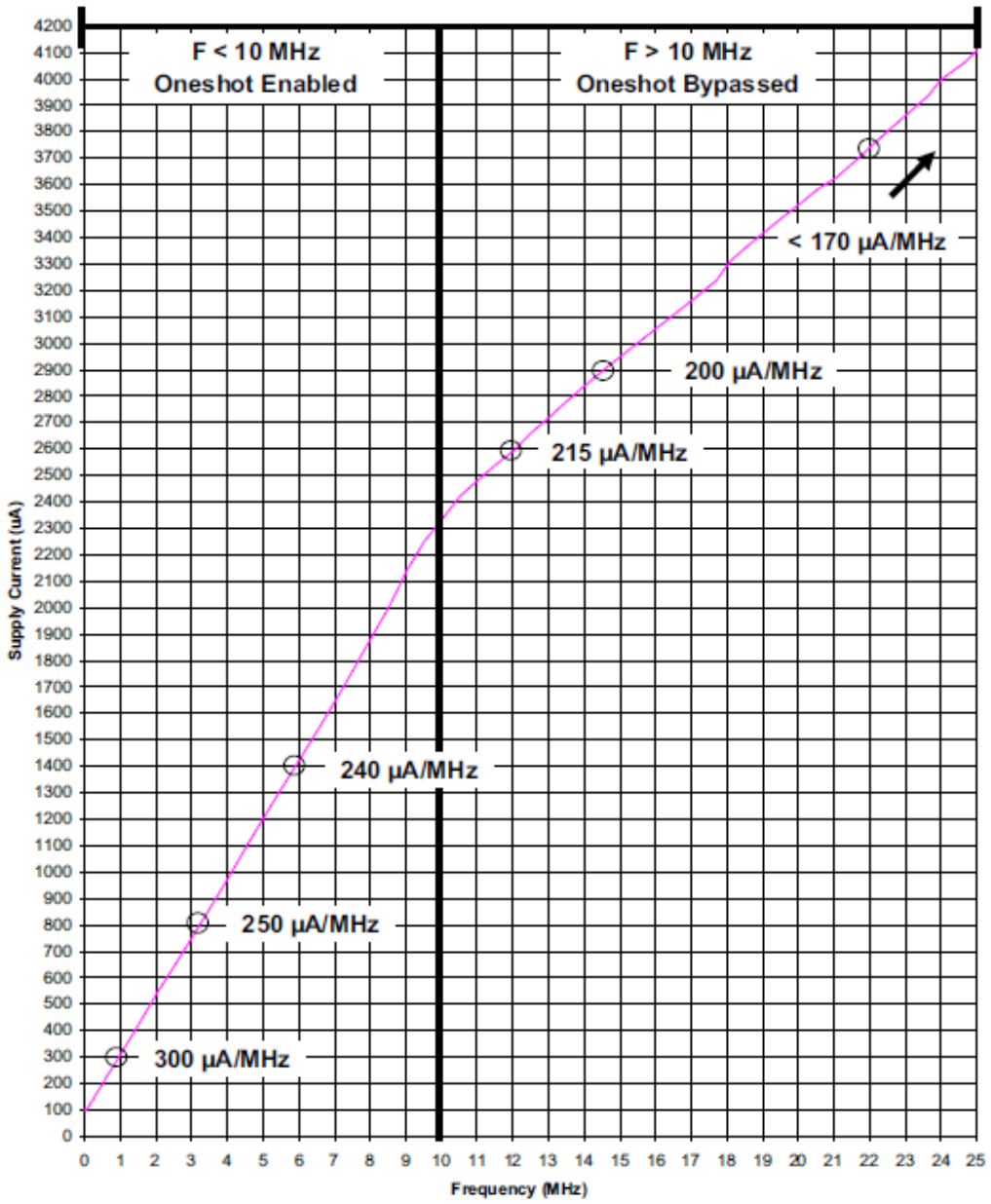


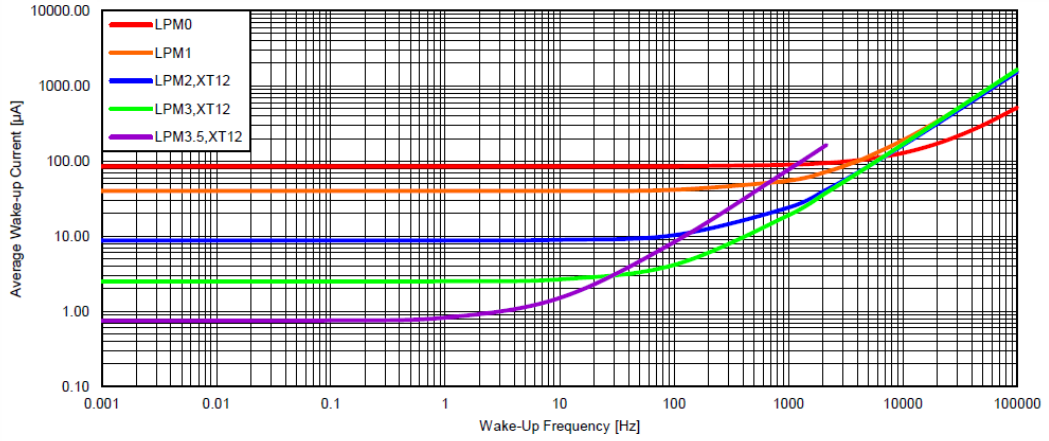
Figure 2.5: Active Mode Current Efficiency.

2.2.2 Wake-up time

Wake-up time refers to the transition from low-power to active modes. For systems with short active windows, the wake-up process consumes a similar amount of current as normal operation. The more functions that are turned off, the less current the chip will consume, but the longer it will take it to wake up.

Another important factor is the oscillator start-up delay, which depends on the clock source as well as other frequency dividers and multipliers in the path. Wake-up time will usually be the limiting factor that determines which low-current mode can be used at a given

time in the application. In order to minimize this issue, some additional peripherals can be used. For instance, an internal voltage regulator that allows to maintain the internal capacitances charged reduces wake-up time by several μs.



NOTE: The average wake-up current does **not** include the energy required in active mode; for example, for an interrupt service routine or to reconfigure the device.

Figure 2.6: Average LPM[2] Currents vs Wake-up Frequency at 25 °C.

Figure 2.6 [9] comes in the sequence of figures 2.3 and 2.4. What this means is that, when the application needs to wake-up very frequently, the best is to keep the device in a higher power mode, allowing power savings in the wake-up process.

2.2.3 Instruction Set Architecture

When evaluating current consumption of a microcontroller, one of the key aspects to take into account is the instruction set. It has an impact on the amount of actual work done per unit of energy consumed. The clocking scheme (ratio between the input system clock and the instruction clock frequencies), cycles per instruction and available instructions, have major impact on current consumption.

The first thought that comes to mind is CISC[3] vs RISC. The CISC architecture goal is to complete a task in as few lines of assembly as possible. This is achieved by building processing hardware that is capable of understanding and executing a series of operations.

On the other hand, RISC architecture is focused in having only instructions that can be executed within one clock cycle. This is achieved by separating the complex instructions with a sequence of simpler instructions.

For instance the ALU (Arithmetic Logic Unit), is one of the most important components of a CPU. The level of complexity has a major impact on current consumption. When an ALU is highly complex with a large number of built-in capabilities, for instance, multiplier/divider units, floating point, registry calculation and so on, that's not always good in what concerns power consumption.

If the application requires very little of these features, a good portion of current is wasted. But if the application requires all these features, and they are unsupported by the ALU, they

[2]Low-Power Modes.
[3]Complex Instruction Set Computers

must be decomposed into simpler instructions, which may require more time to process.

This means that the microcontroller must be adjusted for the application in order to maximize current efficiency.

In order to materialize the above statements, for instance, in some devices, Texas Instruments [10], opted to reduce CPU (ALU) complexity and incorporated an Hardware Multiplier (32x32 bits), taking 1 CPU Cycle[4] to process data, as a peripheral, with no direct support for division. On the contrary, Microchip Technology [5], added hardware support for both multiplication (16x16 bits, 1 CPU cycle) and division (32/16 bits, 18 CPU cycles).

With no direct comparison intended beyond the point of giving two examples, these architectural choices clearly show that these two different devices are useful in different applications.

2.2.4 Data Retention

Most microcontrollers provide two main types of sleep modes. The first is a light-sleep mode, in which the MCU core is stopped, peripherals are disabled, and clock sources are turned off. However, the MCU stays powered up, preserving the contents of registers and SRAM, making the wake-up process faster.

The second is a deep-sleep mode, in which the entire MCU is powered down and SRAM contents are lost. In order to avoid the loss of data, is necessary to save the needed data to Flash or EEPROM memories. When the CPU wakes-up the data needs to be restored in order to resume operation. Unfortunately, due to high write/erase time and current spent on this process, the energy used is substantial.

In order to mitigate some of these problems, another type of memory was introduced in microcontrollers: the Ferroelectric RAM (FRAM) with similar behavior as SRAM except for one important characteristic: it is non-volatile, which means an extra feature to reduce current consumption. FRAM can be used as Flash memory to store the application code and as backup to save data before power down. Although it consumes more current when active, the zero consumption while shutdown allows major power savings.[11]

Table 2.1 shows a short comparison between the previously referred memories [12] [13].

2.2.4.1 Code Execution: Flash vs SRAM

Powering and reading the Flash memory of an MCU is one of the most significant contributions to current consumption. Some devices allow code execution from SRAM. This comes, as expected, with some trade-offs: the first step to run from SRAM is to copy the flash contents to SRAM, which consumes significant amounts of time and current. Another disadvantage is the limited size of the SRAM, which is significantly smaller than flash. This means that, in order to be worthwhile, the functions running from the SRAM must be limited in size or should be computationally-intensive functions.

Another important drawback is the use of a single memory. Since most microcontrollers use two separate buses to read from Flash and SRAM, the limitation of using only one could imply the stalling of the processor in instructions that write back to SRAM.

It can't be forgotten that, applications that switch from active mode to deep-sleep periodically need to restore SRAM contents after each power-on, increasing the waste of power.

[4] Using Direct Memory Access

[5] Write cycle up to 200 MHz

Table 2.1: Comparison between different types of memories.

	SRAM	**FRAM**	**FLASH**	**EEPROM**
Type	Volatile	Non-Volatile	Non-Volatile	Non-Volatile
Write cycle [time]	$<< 125\ ns$ [5]	$125\ ns$	$10\ \mu s$	$5\ ms$
Endurance [cycles]	unlimited	10^{13}	10^5	10^6
Average active current [/MHz]	$< 60\ \mu A$	$110\ \mu A$	$260\ \mu A$	$50\ \mu A$
Flexible code and data partitioning	no	yes	no	no

2.2.4.2 Code Execution: FRAM vs SRAM

As shown in table 2.1, FRAM can work as unified memory with flexible code and data partitioning, meaning that runtime variables can be stored in the same "place" as program data with the drawback explained earlier but with an important advantage: the data saved is kept when the device is shut down. There's still the possibility to use SRAM to complement the FRAM adding the advantage of independent buses.

2.2.5 Proper use of Peripherals

The use of peripheral features in an embedded system is very usual, therefore, maximizing its efficiency is also a good practice to reduce current consumption.

2.2.5.1 Serial Interfaces: UART, SPI and I^{2}C

The serial interface is one of the most used peripherals in an embedded system. Since the protocols used may vary, is necessary to evaluate the best in terms of energy efficiency. Figure 2.7 shows the comparison between three serial interfaces [14].

Parameters	UART		SPI				I^2C			
			RX selected		RX unselected		RX selected		RX unselected	
	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes
Interface implementation in hardware										
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13
Communication:										
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23
Normalized energy[a] μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5
Interface implementation in software										
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23
Communication:										
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6
Normalized energy[a] μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6

Figure 2.7: Energy consumption for evaluated serial interfaces.

The values in this figure (2.7) are normalized and can be used in direct comparison between each other. It can be seen that the most energy-efficient interface is SPI. This is mainly due to the fact that SPI protocol does not have any overhead in data transmission.

2.2.5.2 Analog to Digital Converter

An analog to digital converter consists essentially of a measured voltage across a capacitor with reference voltage circuits, signal amplifiers and buffers, etc. When sampling and converting, all this analog circuits are working and therefore consuming current. When the main objective is to save power, the best option is to run the ADC module at a higher speed than necessary for the application, disabling it between samples and reducing the sampling time to the minimum necessary to maintain measurement accuracy.

In this case some considerations must be taken into account: if the module is being turned off and on, there is a minimum delay before taking a sample. When the source impedance of the analog signal is high the current drawn from the source by leakage can affect accuracy. If the input signal does not change too quickly an extra capacitor must be added to charge the internal holding capacitor, increasing accuracy. At last, putting the microcontroller in a sleep mode also improves accuracy, since the digital noise from the CPU and other peripherals is minimized.

2.2.5.3 DMA and FIFO Buffers

The Direct Memory Access peripheral is mainly used to reduce the CPU overhead, since it allows to perform data transfer tasks without CPU intervention. Since it can work even with the CPU in a sleep mode, it is a great tool to reduce power consumption. This is possible because the DMA module's power consumption is much smaller than the CPU's.

Another important feature is the used of FIFO buffers. These buffers store received data until they are full. Once this happens, an interrupt is fed to the CPU in order to process the data and empty the FIFO. This allows a great power saving because the CPU does not need to handle individually every single transfer.

2.2.5.4 Brown-out Reset

As explained in section 2.2.1 an important peripheral feature is the Brown-out Reset (BOR). While it protects the application from miss-execution due to voltage drop, it also consumes a significant amount of power.

The best way to deal with this issue, without letting the application unprotected from voltage drops, is to have a dynamic BOR. This is achieved by: first, allowing the microcontroller to disable this peripheral in sleep modes, since it's main objective is protect from miss-execution, in sleep modes it is no longer needed when the CPU is not running; second, by using a low-power BOR that ensures that the microcontroller will always have a power-on reset to place it in a known state.

The first approach is not the best, although the CPU is not working does not mean that other peripherals can't be affected by voltage drops. It is always best to have some sort of protection to avoid errors.

2.2.6 Hardware and Software Co-Design

One of the current practices to achieve maximum efficiency is to design simultaneously the hardware and the software [15]. There are some key aspects to take into account:

- Coordination: co-design techniques are used to coordinate the design steps of interdisciplinary design groups;
- Concurrency: Testing hardware (or software) when software (or hardware) is not yet complete.
- Correctness: Complex hardware and software require techniques to not only verify the correctness of each individual subsystem, but also verify their correct interactions after their integration.
- Complexity: today's electronics complexity use co-design techniques to produce correctly working and highly optimized systems.

Several rules apply when designing hardware and software with low-power and maximum efficiency as main goals [3] [16]:

- Minimize operating duty-cycles;
- Minimize leakage currents;
- Maximize impedance in current paths;
- Minimize impedance in high-speed switching paths;
- Use low-power modes whenever possible;
- Use timers to perform delays;
- Use interrupts instead of flag polling;
- Configure unused port pins as outputs and logic level zero - this is dependent on the output's hardware configuration;
- Minimize the use of processing-intensive operations: Modulo, divide, floating point, (s)printf();
- Avoid multiplication on devices without hardware multiplier;
- Use local instead of global variables - this is dependent on the method that most compilers use to pass variables into functions;
- Use 'static' and 'const' for local variables;
- Use pass by reference for large variables;
- Minimize function calls from within interrupt service routines;
- Use lower bits to change program flow;

- Use DMA whenever possible;
- Use count down in loops instead of count up;
- Use unsigned variables for indexing;

2.3 Case Study and Validation

To materialize the above advices and rules, an example provided by Texas Instruments Inc.[17] is analyzed. This is a software that every second takes an ADC temperature sample and the degrees Celsius and Fahrenheit are calculated using floating point operations. The results are then printed and transmitted through the UART.

EnergyTrace ++™ Technology

EnergyTrace™ is an energy-based code analysis tool that measures and displays the application's energy profile and helps to optimize it for ultra-low-power consumption. The energy measurement is made by using a software controlled DC-DC converter to generate the target power supply. The time density of the converter charge pulses equals the energy consumption of the microcontroller. The measurements are made in real-time, which also allows for device-internal state analysis.

ULP Advisor

The Ultra-Low Power Advisor is a software compilation tool that checks source code against the rule list in 2.2.6 that applies to the software part of the application. This allows to identify areas of code with potentially high energy consumption.

When using both tools, it is possible to make software improvements and gather feedback from the behavior of the application.

The ULP Advisor, for this first program version and at the time of compilation, shows the following advices to code optimization:

- Detected *sprintf()* operation(s). Recommend moving them to RAM during run time or not using as these are processing/power intensive;
- Detected floating point operation(s). Recommend moving them to RAM during run time or not using as these are processing/power intensive;
- Detected flag polling using several flags. Recommend using an interrupt combined with enter LPMx and ISR;
- Detected software delay loop using *__delay_cycles*. Recommend using a timer module instead;
- Detected uninitialized ports in this project. Recommend initializing all unused ports to eliminate wasted current consumption on unused pins.
- Detected no uses of low power mode state changes in this project.

The necessary optimization steps with the respective code improvements are shown below. Both complete code versions are shown in appendices A and B.

- *sptrinf()*: Remove. Replace with manual calculation;

  Before:

```
sprintf(resultC,"%.1f Degrees Celcius\r\n", IntDegC);
sprintf(resultF,"%.1f Degrees Fahrenheit\r\n", IntDegF);

UART_print(resultC);
UART_print(resultF);
```

  After:

```
UART_print(rawToAsciiString(IntDegC), 'C');
UART_print(rawToAsciiString(IntDegF), 'F');

char* rawToAsciiString(int16_t input)(...)
```

  In this case, the use of *sprintf()* was replaced with function *rawToAsciiString()*, which makes the manual calculation of the conversion from *float* to *string*.

- Floating point operations: Truncate with integer calculations;

- Divide operations: Replace with multiply and shift whenever possible;

  Before:

```
float IntDegF;
float IntDegC;

IntDegC = (temp - CAL_ADC_12T30) * (85.0-30.0) / (CAL_ADC_12T85 -
    CAL_ADC_12T30) + 30.0;

IntDegF = 9.0*IntDegC/5.0+32.0;
```

  After:

```
int32_t IntDegF;
int32_t IntDegC;

// Temperature in Celsius, multiplied by 10:
IntDegC = ((temp - CAL_ADC_12T30)*10*(85-30)*10 / ((CAL_ADC_12T85 -
    CAL_ADC_12T30)*10) + 30*10);

// Temperature in Fahrenheit, multiplied by 10:
IntDegF = 9*IntDegC/5+320;
```

  The variables and constants used were declared as *int32_t* instead of *float*. The calculation was multiplied by 10 to avoid great losses in precision due to no longer using *floats*.

- Flag polling: Remove whenever possible. Use of interrupts;

- Software delays: Use timers;

- Operation modes: Low power modes whenever possible;

  Before:

```
__delay_cycles(400);                    // Delay for Ref to settle

while(ADC12IFGR1 & ~ADC12IFG30);        // Wait for the conversion to complete
while(ADC12CTL1 & ADC12BUSY);           // ADC12 Control Register & Busy bit
while(!(UCA0IFG & UCTXIFG));            // Wait until TX buffer ready
while(UCA0STATW & UCBUSY);              // USCI A0 Stat Reg & Busy Flag
```

  After:

```
// Configure Timer
TA0CTL = TASSEL__ACLK | MC__UP | TACLR; // ACLK, up mode, clear timer.
TA0CCR0 = 131;                          // ~0.4ms
TA0CCTL0 |= CCIE;                       // Cap/comp interrupt enable

__bis_SR_register(LPM3_bits | GIE);     // LPM3. Wait for Ref to settle

TA0CCR0 = 0x8000;                       // Change timer delay to  ~1 sec.

while(1)
{
  __bis_SR_register(LPM3_bits | GIE);   // Enter LPM3, wait for ~1sec
      timer in a low-power mode

  ADC12CTL0 |= ADC12ENC | ADC12SC;      // Sampling and conversion start
  __bis_SR_register(LPM3_bits | GIE);   // Wait for conversion to complete
       entering a low-power mode.

  (...)
}

#pragma vector=TIMER0_A0_VECTOR   // Exits Low-power mode
__interrupt void TIMER0_A0_ISR(void)(...)

#pragma vector = ADC12_VECTOR     // ADC interrupt service routine
__interrupt void ADC12_ISR(void)(...)

#pragma vector=USCI_A0_VECTOR     // Sends data through UART
__interrupt void USCI_A0_ISR(void)(...)
```

  The flag polling cycles, *while(...)*, were replaced by timer and interrupt service routines configuration. While waiting for the peripherals, the CPU is placed in a low-power mode.

- Floating port pins: Declare output with logic level low;

  Before:

```
// Configure used port pins
P2SEL1 |= BIT1 | BIT0;                  // Configure UART pins
P2SEL0 &= ~(BIT1 | BIT0);               // Configure UART pins
```

  In this case, the absence of configuration means the port is left floating, which means its output/input state is undefined. To avoid this, for the MSP430 family of devices,

the ports must be configured as inputs or outputs driven low. These configurations highly depend on the ports' hardware and can vary depending on the manufacturer of the microcontroller.

After:

```
// Configure Port Pins as Output Low. Clear all port interrupt flags.
PAOUT = 0; PBOUT = 0; PJOUT = 0;
PADIR = 0xFFFF; PBDIR = 0xFFFF; PJDIR = 0xFF;
PAIFG = 0; PBIFG = 0;

// Configure used port pins
P2SEL1 |= BIT1 | BIT0;                    // Configure UART pins
P2SEL0 &= ~(BIT1 | BIT0);                 // Configure UART pins
```

- Loop count: Count down in loops.

  Example:

```
for (i = 16, bcd = 0; i; i--)
(...)
for (i = 4; i > 0; i--)
(...)
```

  As previously explained, using decrementing variables in loops is more efficient due to the presence of native instructions to make a direct comparison to zero.

- Clocks: Deactivate unnecessary clock sources.

  Before:

```
(...) // No configuration. All clocks running
```

  After:

```
// Configure Clock System
CSCTL0_H = 0xA5;                          // CS password
CSCTL2 = SELA__LFXTCLK;                   // ACLK sourced from LFXT
CSCTL3 = DIVA__1;                         // No division
CSCTL4 |= LFXTDRIVE_3 | SMCLKOFF | VLOOFF; // Highest crystal drive
    setting.
CSCTL4 &= ~LFXTOFF;                       // Turn on LFXT
```

  By default, after a reset, all clock sources are enabled, which means a great increase in power consumption. Therefore, like the I/O ports, all clock sources must be configured.

Using EnergyTrace++™ is possible to track relative energy consumption and compare different optimization procedures. Figures 2.8, 2.9 and 2.10 show the improvements from the inefficient version to the most efficient.

From figure 2.8, the first noticeable improvement is the overall system's energy consumption reduction (*Delta Energy*): 89.5%. This is achieved in great part by the transition to low-power modes when the CPU is idle (*Active Mode* → *LPM3*) . Figure 2.8 also shows an increase in time and energy spent in low-power modes of 83%. These modes deactivate the main system clocks (ACLK, MCLK) and consequently deactivate all peripherals sourced by these clocks, saving energy.

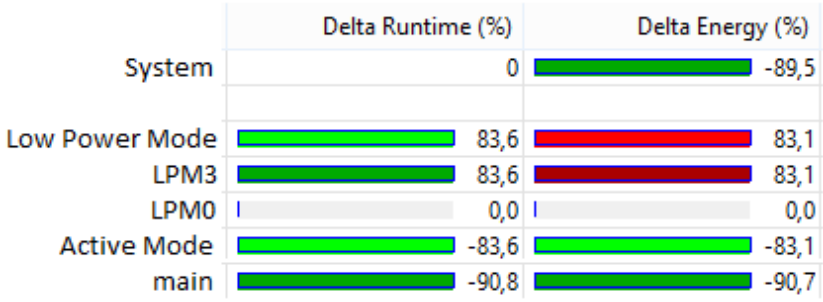


Figure 2.8: Operation modes and *main* function runtime and energy comparison.

The complete disregard of optimization and low-power design techniques and advices, highlighted by the inefficient version of the software, clearly shows the extra energy consumed to achieve the exact same goals and results.

The need for optimization becomes even more evident when verifying code size after compilation. The *Very Inneficient* version takes 14.294 bytes and the *Optimized* takes only 1.618 bytes.

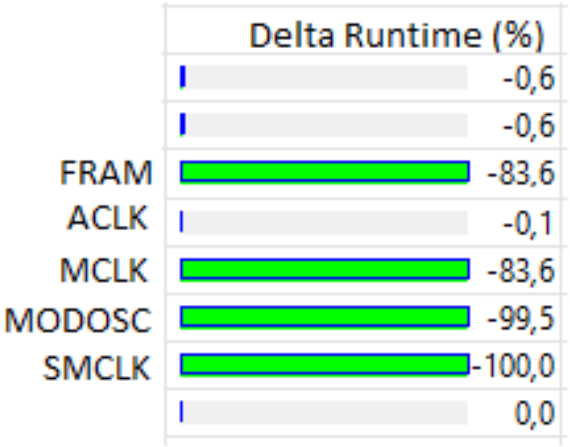


Figure 2.9: Peripherals and system clocks runtime comparison.

Figure 2.10 shows the final obtained results. The level of improvement from one version to another is ultimately noticeable by the duration of a CR2032 battery running both applications.

One important note is necessary to take into account: The need to know how the compiler works. A programmer could try to make all optimizations listed above and still does not achieve the lowest power consumption possible, since the program compiler makes code optimizations not always good in terms of power consumption.

2.4 Chapter Remarks

This chapter started with a brief introduction to power categories in order to give visibility to important factors that cannot be overlooked when designing a low-power system. It was

EnergyTrace™ Profile	Optimized	Very Inefficient
Name	Live	Live
System		
Time	30 sec	30 sec
Energy	2,39 mJ	133,34 mJ
Power		
Mean	0,10 mW	4,66 mW
Min	0,049 mW	2,433 mW
Max	0,235 mW	7,815 mW
Voltage		
Mean	3,59 V	3,59 V
Current		
Mean	0,03 mA	1,30 mA
Min	0,014 mA	0,678 mA
Max	0,065 mA	2,175 mA
Battery Life	CR2032: 344,7 day (est.)	CR2032: 6,2 day (est.)

Figure 2.10: Optimization results.

also theoretically demonstrated that running as fast or as slow as possible is not always better - a balance must be achieved.

Another topic covered by this chapter is the proper use of peripherals to run the appropriate configurations or to provide extra tools to save power.

Several low-power hardware and software techniques were presented and its impact was demonstrated when used properly on a real system. Although several configurations make the program more difficult to read or even to code due to the low-level programming, that is sometimes necessary to optimize the system at all levels.

Chapter 3

Energy Harvesting

The concept of energy harvesting in sensor and microcontroller systems is mostly related to the process of converting ambient or waste energy into electrical energy in order to power those devices.

The development of energy harvesting has been driven mainly by the proliferation of autonomous wireless electronic systems. This demand for energy efficient devices increased the research and development of self powered systems using energy harvesting with the perpetual system as main goal.

The power generated by this process is in the range of micro to milliwatts and is highly dependent on local conditions. When designing energy harvesting systems, several technical questions and challenges must be overcome, namely: availability of the energy source; time variation; multiple sources exploit and cost-effectiveness. The implications of each method should be properly measured before committing to an approach. Why choose energy harvesting? For instance, one of the most critical applications is the pacemaker. Linear and nonlinear piezoelectric devices are introduced to continuously recharge the batteries of the pacemakers by converting the vibrations from the heartbeats to electrical energy. The power requirement of a pacemaker is very low. However, without energy harvesting, after few years, patients require another surgical operation just to replace their pacemaker battery [18] [19].

3.1 System Architecture

Usually, the outputs of an energy harvester are not suitable for direct use to power the devices. Depending on the nature of the harvester, the characteristics of voltage and current can vary, namely the phase, amplitude and frequency of the AC waveforms or the magnitude of the DC level.

In order to power the electronics it is necessary to properly process the harvester's output. Furthermore, it is not expected that the harvester would be able to give enough continuous energy to power the electronic devices, therefore an energy storage unit is necessary. The storage of energy can be made with a super capacitor or with a rechargeable battery.

3.2 Energy Sources

The choice of energy source and method of implementation is largely defined by the application. There can be a fundamental link between the energy source and the design of

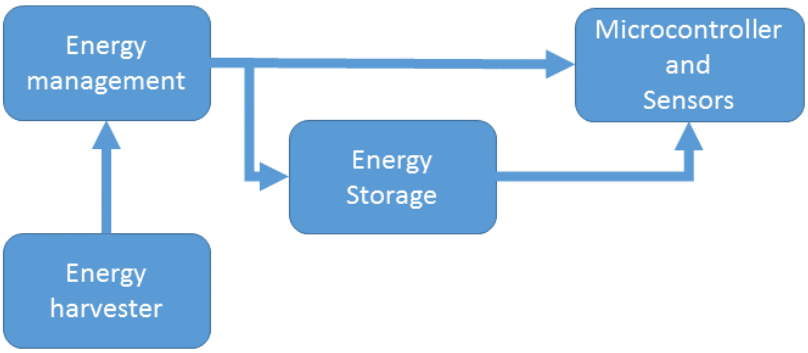


Figure 3.1: Energy harvesting system architecture

the harvester. Several sources can be identified [20]:

Motion, vibration or mechanical energy

Kinetic energy harvesters are typically inertial spring-mass systems [21]. Electrical power is extracted by employing piezoelectric, electromagnetic and electrostatic transduction systems. Piezoelectric generators use active materials that generate an electrical charge when stressed mechanically. Electromagnetic generators operate based on Faraday's law of induction, which arises from the relative motion of a conductor moving through a magnetic flux. Finally, electrostatic generators utilize the relative movements between electrically isolated charged capacitor plates to generate energy. In figure 3.2, the regenerative brake system widely used in electric vehicles is shown. It converts kinetic into electrical energy which is stored in a battery.

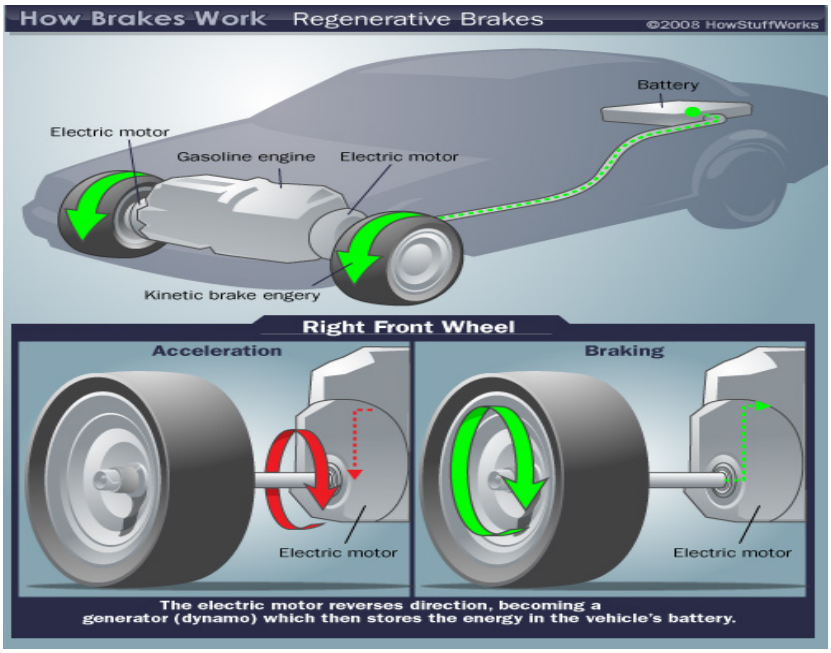


Figure 3.2: Regenerative brake diagram. src: HowStuffWorks.

Electromagnetic (RF)

Ambient energy density from electromagnetic waves is diverse in frequency and the lowest among ambient renewable energy sources [22]. They are originated from radio waves from transmission cell towers. Due to this, there is the need to have both multi-band/broadband capabilities of the realized wireless electromagnetic harvester and relatively high harvesting efficiency for possible applications in powering remote sensors. The accumulation of electromagnetic energy from various frequencies will increase the energy density of the harvester and make it a viable alternative to other existing techniques [23]. Figure 3.3 [24] shows a multi-band planar antenna for energy harvesting.

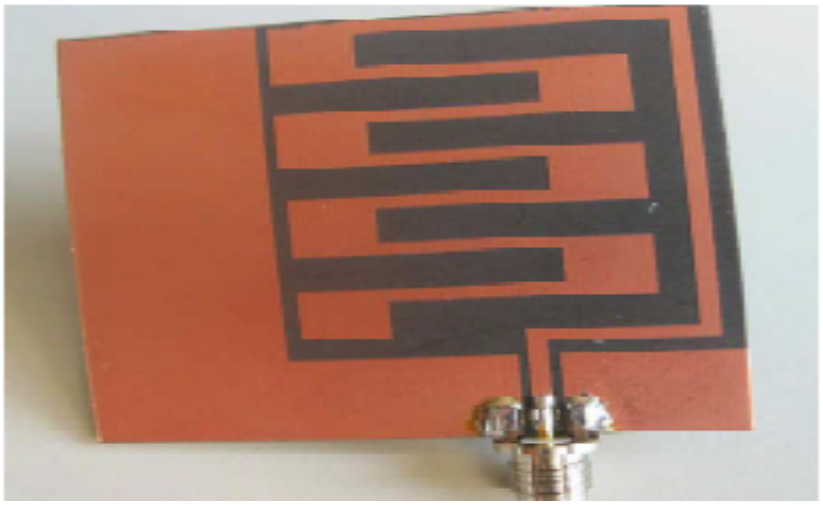


Figure 3.3: Multi-band planar antenna.

Thermal

Thermal energy harvesters are based on the Seebeck effect: when two junctions, made of two different conductors, are kept at different temperatures, an open circuit voltage is generated between them. This is the principle of the thermocouple.

The thermal harvester is based on the thermopile. This is a device formed by a large number of thermocouples. It will be explained in the next chapter.

Figure 3.4 shows a energy harvester based in a heat sink. When the conversion efficiency is not the most relevant issue in thermoelectric applications, waste heat recovery is a good opportunity due to its simplicity and reliability [23].

Micro water flow

The principle of operation is the same as dams use to generate power but at a much smaller scale. The water pressure present in household pipes is used to generate energy using hydro-generators.

The energy harvester together with a power management circuit and storage can be used to supply power to automatic faucets or other small electronics [25] as exemplified in figure 3.5.

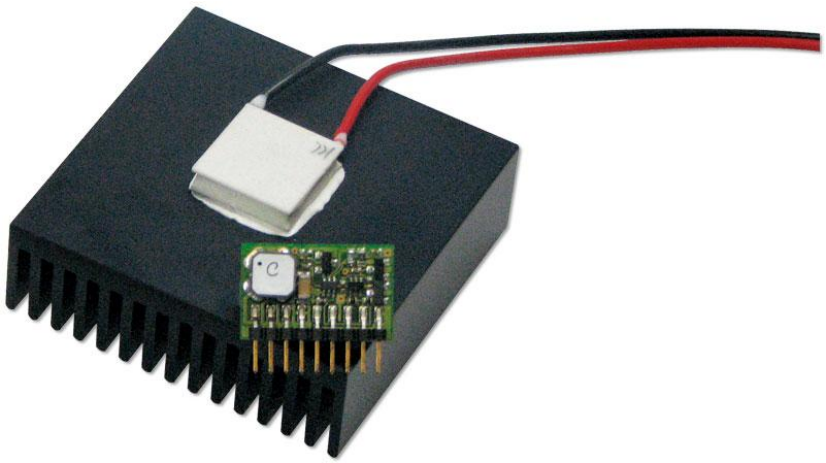


Figure 3.4: Thermal energy harvesting in a heat sink. src: rtcmagazine.com

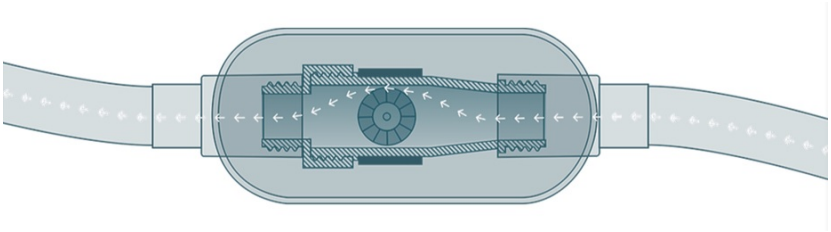


Figure 3.5: Micro water turbine. src: element14.com

Solar and light

Natural and artificial light can be collected and converted to usable energy using photovoltaic panels. The intended scale in this case is comparable to solar-powered calculators or watches.

Usually, the power output isn't very large, but because it is continuous the total energy collected is normally more than sufficient to power small devices such as the one shown in figure 3.6.

Figure 3.6: Solar powered mini car. src: inhabitat.com

Biological

Power may be recovered passively from body heat, breathing, blood pressure, arm motion, walking or other activities such as pedaling. Energy harvesting from biological sources follow mostly the principles presented in subsection 3.2.

Figure 3.7: Energy harvester human motion concept. src: Krupenkin, T. & Taylor, J. A. - kurzweilai.net

3.3 Chapter Remarks

When choosing and deploying an energy harvester, it is necessary to take the appropriate measures to increase its efficiency. Additional circuitry is necessary to implement impedance matching between the harvester and load electronics, energy storage and voltage regulation.

Each energy harvester has its own model for impedance matching. Achieving this, allows for maximum energy transfer. The overhead of the power processing stages must be kept as low as possible in order to maximize the efficiency of the harvester. Since in these systems the power available is in the μW range, leakages and other power dissipations are critical and must be as low as possible.

Chapter 4

Infrared Detection

In order to understand the issues associated with infrared detection, is necessary to analyze three constituents of the process: the radiation source, propagation medium and the receiver.

In contrast with the visible domain, where ambient perception is mainly made through diffracted, scattered and reflected light, thermal infrared radiation is emitted by the objects themselves due to its own temperature.

4.1 Radiation Sources

According to the Planck's law that describes the electromagnetic radiation emitted by a blackbody in thermal equilibrium at a definite temperature, the spectral radiance, per unit wavelength, can be defined as: [26] [27]:

$$B_\lambda(\lambda, T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{hc/\lambda k_B T} - 1} \qquad [\frac{\mathrm{W}}{\mathrm{sr}\ m^3}] \tag{4.1}$$

where k_B is the Boltzmann constant, h the Planck constant, λ is the wavelength, T is the temperature in Kelvin and c the speed of light in the medium. The thermal radiation emitted by other objects is derived through a parameter called emissivity, $\epsilon < 1$.

The emission of thermal radiation is not uniform over the whole electromagnetic spectrum. Around 70 % of the energy emitted is concentrated between $1/2\lambda_{peak}$ and $2\lambda_{peak}$. The λ_{peak} is the wavelength at which the energy radiated is the highest. This is derived form the blackbody temperature by Wien's law [27].

$$\lambda_{peak} = \frac{2898}{T} \tag{4.2}$$

where λ_{peak} is in micrometers and T in Kelvin. In the case of the human body, with a surface temperature of 35 ^oC = 308 K the peak wavelength of the radiated energy is 9.4 μm, in which 70 % is between 4.7 μm and 18.8 μm. It is also known that the energy of a electromagnetic wave is given by (4.3).

$$E = \frac{hc}{\lambda} = \frac{1.241}{\lambda} \qquad [eV] \tag{4.3}$$

where h is the Planck constant, c is the speed of light in the propagation medium and λ in micrometers. For the example above, $E = 0.13\ eV$.

Infrared radiation is placed next to the visible component in the electromagnetic spectrum with wavelengths from 0.75 μm to 1000 μm with the following characteristics:

- Invisible to human eyes;
- Small energy;
- Long wavelength;
- Emitted from all kinds of objects;

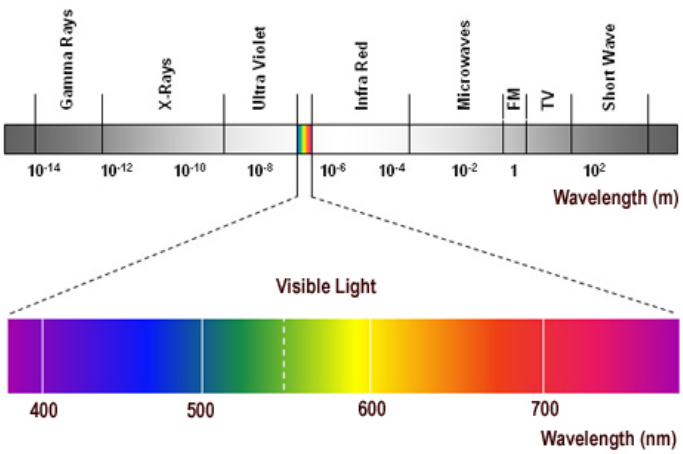


Figure 4.1: Electromagnetic Spectrum. src: pro-lite.co.uk

4.2 Propagation Medium

The most common propagation medium in infrared applications is the air/ atmosphere. Since it is an open medium, it is highly susceptible to disturbances from all sources of radiation. For instance, visible light, as can be seen in figure 4.1 and by (4.3) has greater energy than infrared radiation making it a common disturbance source to take into account in infrared detection, depending on the sensitivity of the receiver at different wavelengths.

4.3 Infrared Receivers

There are two main types of radiation detectors: thermal and photonic [28].

Thermal detectors are sensitive to the energy carried by the object in which the detector heats up when absorbing incident radiation and the increase in temperature causes a change in the physical characteristics of the material.

Photonic detectors use electrical properties of semiconductors. They rely on the direct interaction between photons and electrons. The incident radiation provides energy that allows charges to change band gaps provided that the incident energy is greater than the forbidden gap. This is illustrated in figure 4.2.

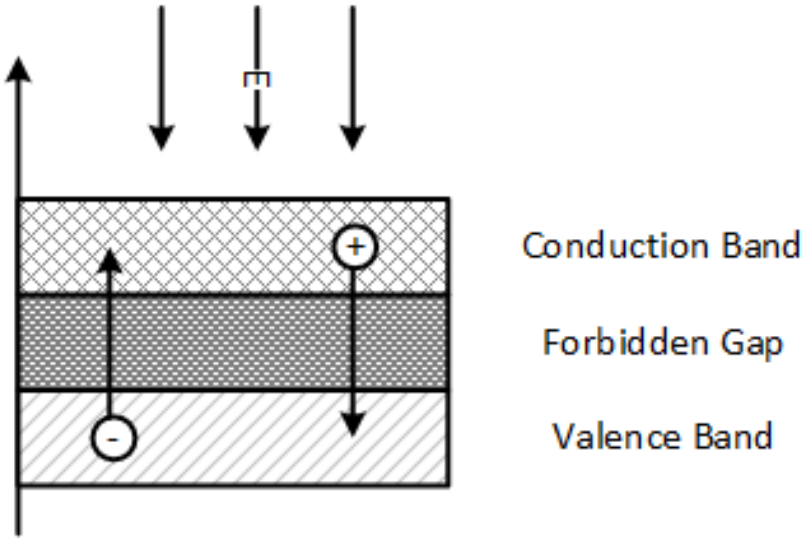


Figure 4.2: Energy bands diagram.

4.4 Detector Main Characteristics

As with all systems transmitting information, a detector causes a spontaneous degradation of the signal. Noise is a random fluctuation in the electrical output from a detector and must be minimized to increase the performance sensitivity of an infrared system.

The statistical properties of the radiation are represented by the probability distribution laws of random processes. The emission and absorption of photons happens in packets, each of these packets containing a different number of photons, with the time interval which separates them itself being a variable.

Noise Sources

According to [29], the most common noise sources are shot, pink, generation - recombination and noise due to temperature changes. These are all a function of the detector type, area, bandwidth and temperature. This distribution is illustrated in figure 4.3.

An important characteristic of infrared detectors is the sensitivity, that is the ability to respond to weak signal variations and is set by the statistical nature of the radiation to which it responds.

Random noise sources appear, as stated above, in voltage, current or power fluctuations. These random processes are defined by its mean, variance, standard deviation and power spectral density.

Considering the voltage as the output of the detector, the random noise waveform $v_n(t)$ and the corresponding probability density function of the random process, the following can be defined [30]:

Mean

$$\bar{v}_n(t) = \frac{1}{T}\int_0^T v_n(t)dt \quad [V] \tag{4.4}$$

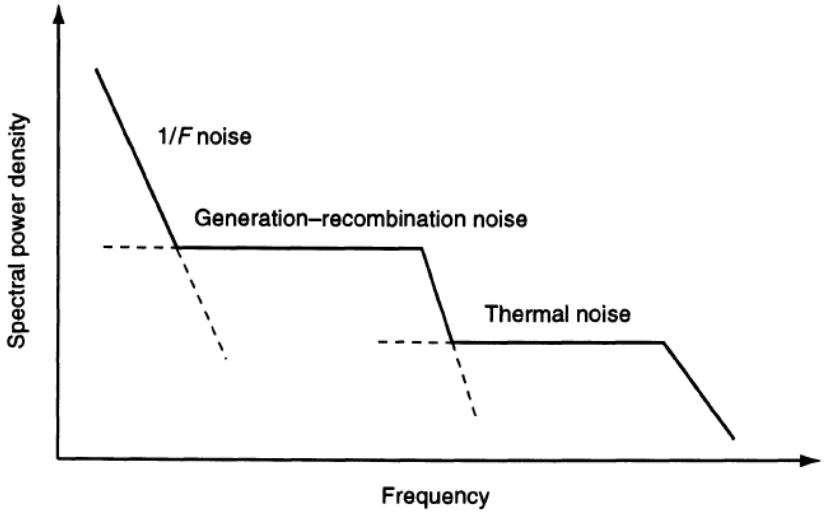


Figure 4.3: Spectral distribution of electrical noise sources in a semiconductor

Variance

$$\bar{v_n^2} = \frac{1}{T}\int_0^T [v_n(t) - \bar{v}_n]^2 dt \quad [V^2] \tag{4.5}$$

Standard Deviation

$$v_{rms} = \overline{\Delta v_n} = \sqrt{\frac{1}{T}\int_0^T [v_n(t) - \bar{v}_n]^2 dt} \quad [V] \tag{4.6}$$

where T is the time interval. The standard deviation represents the rms noise of the random variable.

The contribution of different noise sources is calculated through its power (variance):

$$v_{rms,total}^2 = v_1^2 + v_2^2 + ... + v_n^2 \tag{4.7}$$

Power Spectral Density

The time-average autocorrelation function is defined in (4.8) and is the measure of how fast the waveform changes in time.

$$c_n(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} v_n(t)v_n(t+\tau)dt \tag{4.8}$$

In frequency domain the noise distribution can be defined by the power spectral density and provides a measurement of frequency distribution of the power.

$$N(f) = \mathfrak{F}\{c_n(\tau)\} = \int_{-\infty}^{+\infty} c_n(\tau)e^{-j2\pi f\tau}d\tau \tag{4.9}$$

To obtain the average power of the random voltage waveform is necessary to integrate the power spectral density over the entire range of definition.

$$c_n(0) = \int_{-\infty}^{+\infty} N(f)df = \int_{-\infty}^{+\infty} v_n^2(t)dt = \bar{v}_n^2(t) \tag{4.10}$$

Most calculations when considering noise are made over the white noise region, where the spectral density is flat enough in a wide range of frequencies.

Generation-Recombination Noise

Generation-recombination noise is associated with the spontaneous variation in the number of charge carriers in a semiconductor. This is of either thermal origin or photonic origin.

The variance of the noise current is photo-conductors is given by:

$$\overline{(\delta I)^2} = 4qGI\Delta f \tag{4.11}$$

where G is the detector gain and q the electron charge.

Thermal Noise

Another noise source caused by the thermal agitation of charge carriers and occurs is the absence of electrical bias as a fluctuating rms voltage or current.

$$v_n = \sqrt{4k_BTR\Delta f} \tag{4.12}$$

where k_B is the Boltzmann constant, T is the temperature in Kelvin, R is the value in Ohms and Δf is the bandwidth in Hz.

The noise power dissipated in a matched load, that is, a value equal to the internal resistance R is:

$$P_n = k_BT\Delta f \quad [W] \tag{4.13}$$

And the spectral noise power density:

$$P_f(f) = k_BT \tag{4.14}$$

1/F or Pink Noise

This kind of noise only appears at low frequencies ($f < 1$ kHz) where it is superimposed on the previous noise sources. This appears in the detectors due to the DC bias currents flowing in the detector material.

This can be partly excluded by cutting the system DC response using a high pass filter with the appropriate cutoff frequencies.

Shot Noise

Shot noise can come from a current having a photonic origin. This means it can be related to interference due to ambient light and with power proportional to the optical power impinging the detector [31].

The dc current is viewed as the sum of many short and small current pulses. This type of noise is practically white, considering the spectral density of a single narrow pulse.

Noise Equivalent Bandwidth

The noise equivalent bandwidth is defined as the bandwidth of a perfect rectangular filter that possesses constant power-gain distribution between the lower and upper frequencies and zero elsewhere.

$$\mathrm{NE}\Delta f = \frac{1}{G^2(f_0)} \int_{-\infty}^{+\infty} |G(f)|^2 df \quad [\mathrm{Hz}] \tag{4.15}$$

where $G(f)$ is the power gain as a function of frequency and $G(f_0)$ is the maximum value of the power gain. This expression is valid when considering white noise.

Noise Equivalent Power

The noise equivalent power represents the detection ability of a detector and is expressed in the quantity of the incident light in which the signal-to-noise ratio becomes unity. A smaller NEP corresponds to a more sensitive detector.

4.5 Existing Detectors and Limitations

When considering common applications, three detector types are usually used: passive, active and thermopile sensors.

4.5.1 Passive Sensors

Actual passive infrared sensors, used for motion detection, rely on two or more separate halves and a infrared filter lens. These halves are wired up so they cancel each other, since the objective is to detect change and not average infrared levels. In figure 4.4, the working principle is illustrated [32]. The signal output from a detector is usually small and therefore an amplification is necessary.

Passive sensors only rely on a detector and degradation of the signal can have its cause in physical imperfections of the sensor material and/or limitations tied to the physical principles.

To converge or focus infrared radiation, several lenses made of different materials are used according to the target wavelength. This is the typical form of filtering against disturbance sources.

These sensors are very sensitive to rapid temperature and light changes. A rapid change due to a breeze from an open window or even a small shadow can cause false triggering. Furthermore, the orientation of the sensor is extremely important. In figure 4.5 is illustrated a typical implementation for motion detection.

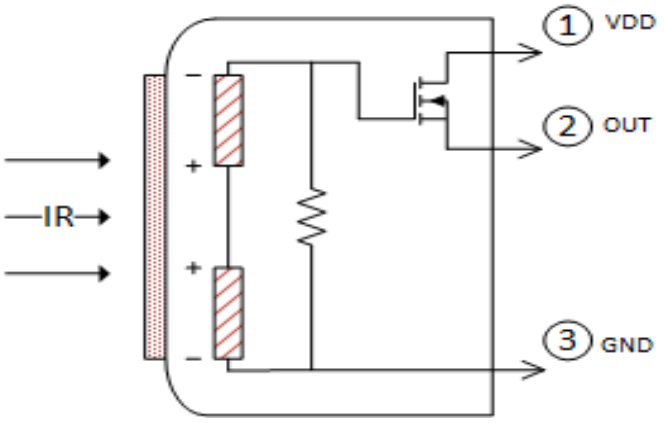


Figure 4.4: Passive infrared sensor working principle.

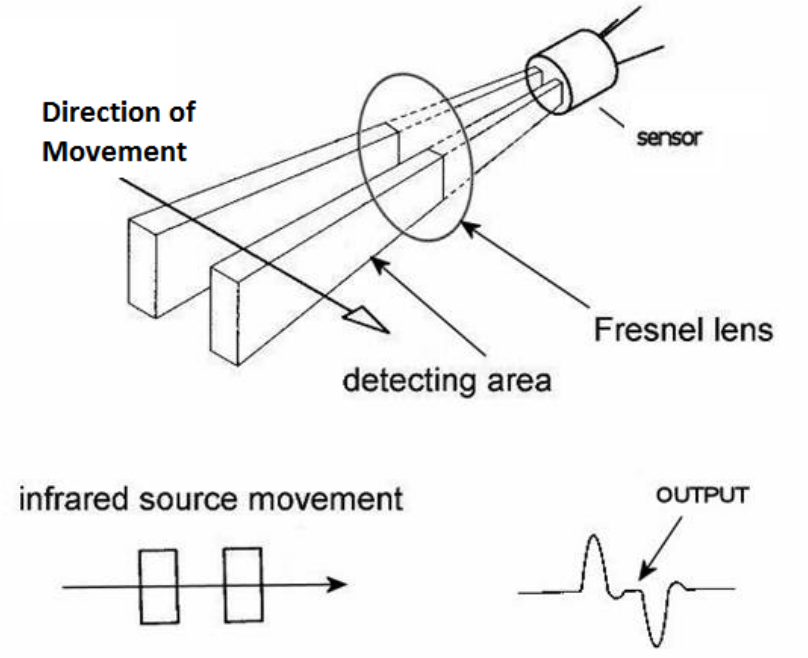


Figure 4.5: Passive infrared sensor working principle. src: adafruit.com

Since these sensors rely on differential readings, if the movement is slow and in the same orientation (vertical or horizontal) as the sensor halves, there is no differential change and therefore no output is generated.

4.5.2 Active Sensors

On the other hand, active infrared sensors, emit and modulate a signal, typically at 40 kHz, and rely on a receiver that can only detect infrared pulses of the same frequency, usually by implementing a bandpass and infrared filters [33].

The operation of the system starts with a 40 kHz impulse generator that is fed to the infrared led. A weaker signal but with the same frequency is reflected from an possible obstacle

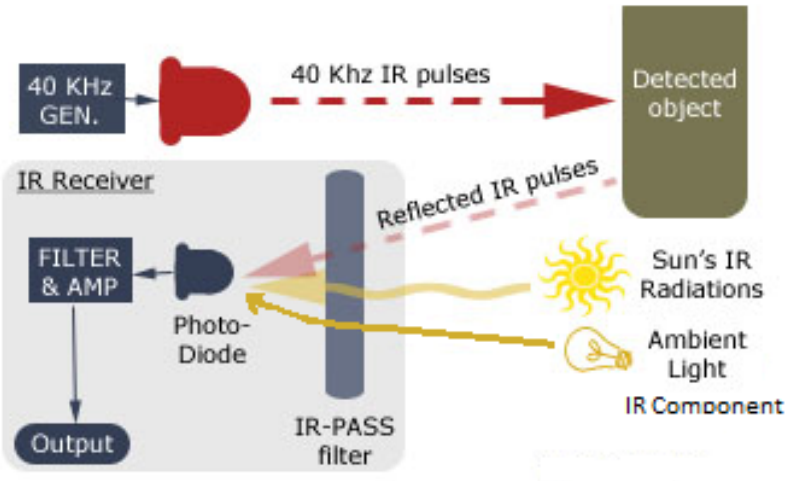


Figure 4.6: Modulated active method of operation. src: adafruit.com

to the infrared receiver. This signal passes through a filter that blocks all other sources of light not belonging in the infrared region. Since all other infrared sources still pass through the filter as noise an additional bandpass filter is necessary to select only the 40 kHz signal.

Although the emitted signal is modulated, the light coming from interferences can overlap the reflected signal making it undetectable. The main weakness of infrared remains: the performance highly depends on the shape, size and color of the reflective object and ambient interference sources.

4.5.2.1 Active Sensor: TSAL6100 & TSSP58P38

An example of active infrared sensors, is the combination of the TSAL6100 & TSSP58P3 (emitter and receiver respectively).

TSAL6100 [34] is an infrared, 940 nm emitting diode in $GaAlAs$ multi quantum well (MQW) technology with high radiant power and high speed molded in a blue-gray plastic package.

Table 4.1: TSAL6100 main characteristics.

Radiant Intensity	170 mW/sr
Angle of half intensity	$\varphi = \pm 10$
Wavelength	940 nm

The TSSP58P38 [35] is a compact infrared detector module for proximity sensing applications. It receives 38 kHz modulated signals and has a peak sensitivity at 940 nm. The length of the detector's output pulse varies in proportion to the amount of light reflected from the object being detected. The visible light is suppressed by an IR filter and includes the photo-detector and preamplifier in one package.

With the circuit shown in figure 4.9, with a 38 kHz square wave and a variable duty cycle is possible to obtain the graph in figure 4.10.

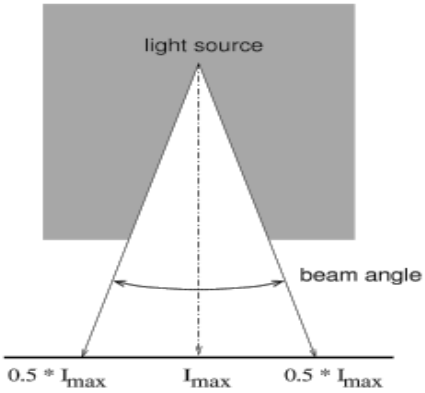


Figure 4.7: Angle of half intensity

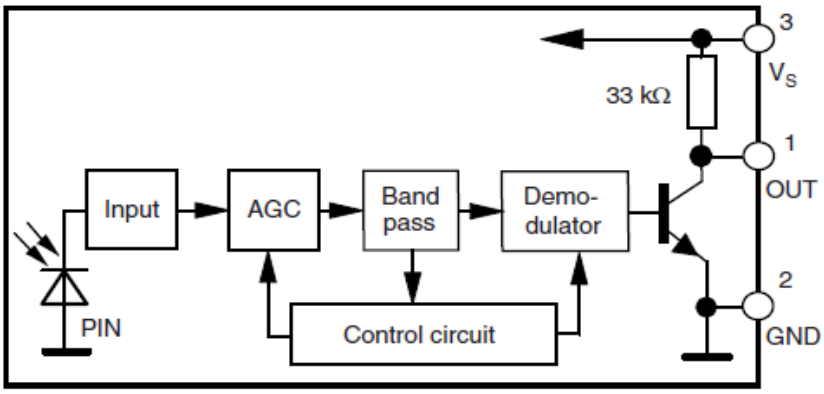


Figure 4.8: TSSP58P38 block diagram.

Table 4.2: TSSP58P38 main characteristics.

Supply Current	[0.55 ; 0.9] mA
Supply Voltage	[2.5 ; 5.5] V
Angle of half receiving distance	$\varphi = \pm 45^{o}$
Wavelength	940 nm

This behavior is explained by the spectral analysis of the PWM signal with 50% and 40% duty cycle (figures 4.11 and 4.12). Since the infrared receiver has its peak responsivity at 38 kHz, only the magnitude of the emitted signal at this frequency is relevant. That is, the bigger the frequency spreading the less efficient the system will be.

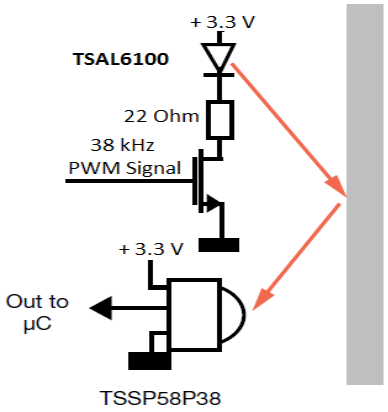


Figure 4.9: TSAL6100 and TSSP58P38 typical application circuit.

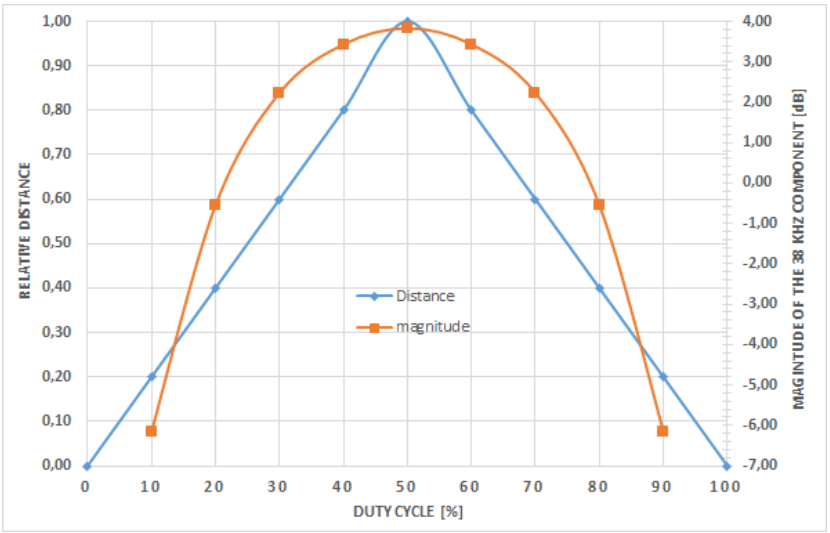


Figure 4.10: Magnitude graph of TSAL6100 and TSSP58P38 sensors.

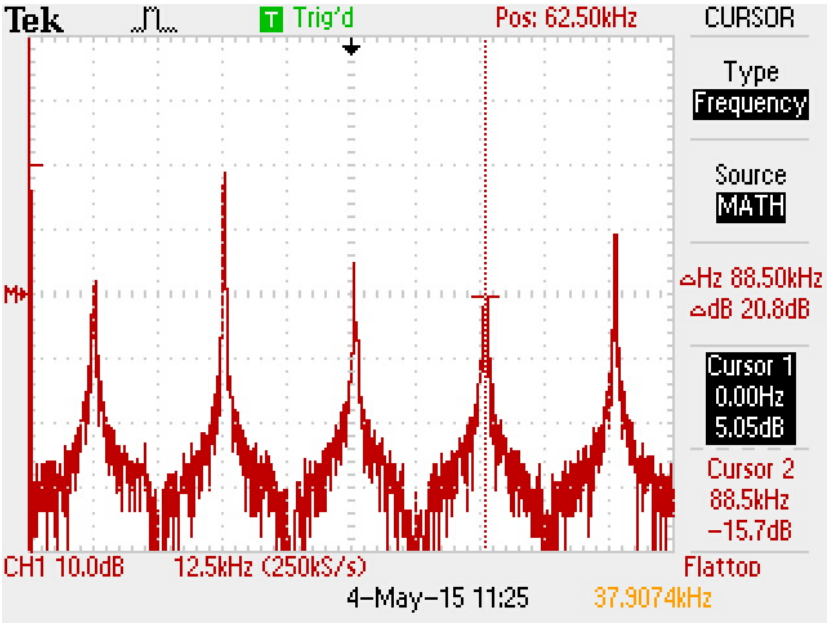


Figure 4.11: Spectral analysis of the PWM signal with 50% duty cycle.

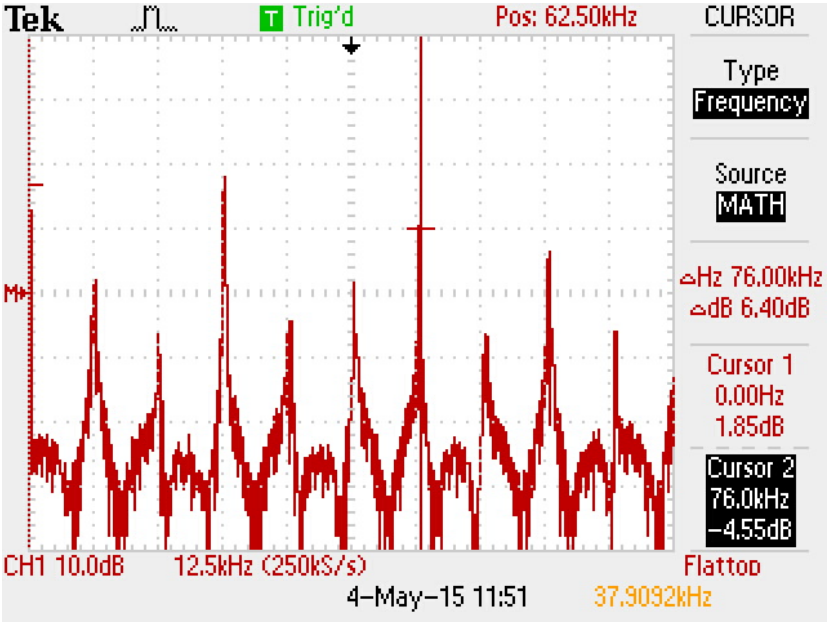


Figure 4.12: Spectral analysis of the PWM signal with 40% duty cycle.

4.5.2.2 Active Sensor: SHARP 2Y0A21 F 9Y

The SHARP 2Y0A21 F 9Y [36] is a distance measuring sensor unit, composed of an integrated combination of a position sensitive detector (PSD), an IR LED and signal processing circuit, with analog output type.

The distance detection is not easily influenced by the variability of the reflectivity of the object and by environmental temperature changes because of the adoption of the triangulation method. This device outputs the voltage corresponding to the detection distance.

Table 4.3: SHARP 2Y0A21 F 9Y Main characteristics.

Distance range	[10 ; 80] cm
Current Consumption	30 mA
Supply Voltage	[4.5 ; 5.5] V

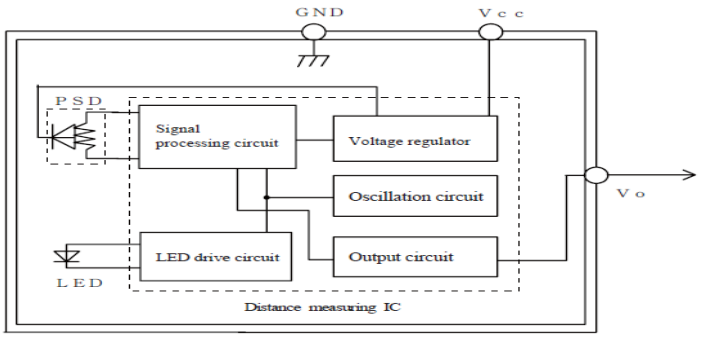


Figure 4.13: Block Diagram

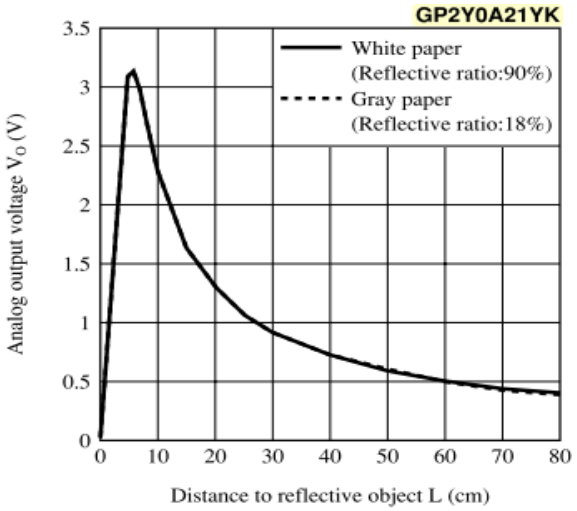


Figure 4.14: Output voltage vs distance to reflective object.

The PSD is a silicon component that operates on the principle of the photoelectric effect, in

which light energy is turned into electrical energy. The top of the PSD contains photosensitive material, and attached to the left and right ends of the device are output current leads. For a focused beam, such as one produced by a laser, the ratio of the output currents are proportional to the location at which the beam strikes the PSD; equal currents indicate that the beam has struck at the middle of the PSD. This dual output system eliminates the power of the reflected beam from the equation of position. This is necessary as every surface has a different reflection coefficient. When the beam strikes an object it is reflected back towards the sensor and into a focusing lens which directs the reflected beam onto the PSD. The incident angle of the reflected beam is determined by the distance from the sensor to the reflecting object; the farther away the reflecting object is, the slighter the angle.

Figure 4.15 has the purpose of corroborating part of figure 4.14, where it is shown the DC voltage present when no object is detected as well as the voltage around a distance of 10 cm.

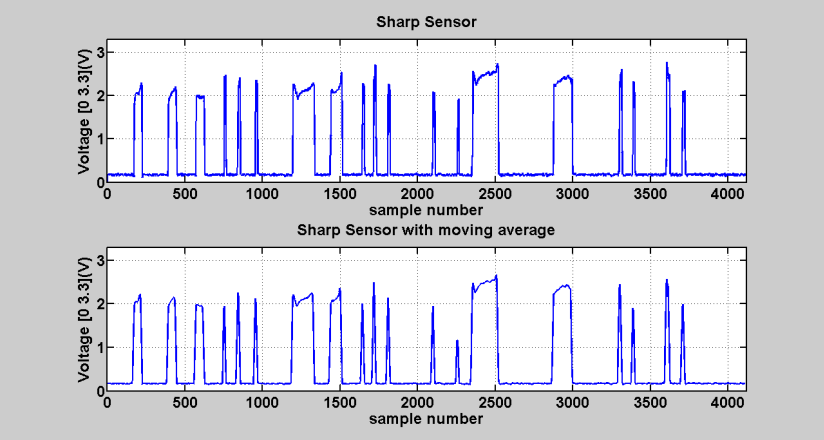


Figure 4.15: Detection results for a human hand at 10 ± 2 cm . a. Obtained signal; b. moving average with 12 samples.

4.5.3 Thermopile Sensors

A thermopile is an electronic device that converts thermal energy into electrical energy. It is composed of several thermocouples usually connected in series. The thermopile does not respond to absolute temperature, but instead generates an output voltage proportional to a local temperature difference.

Most common thermopile sensors are contactless meaning that is not necessary to make physical contact with the object. The thermopile absorbs the infrared energy emitted from the object being measured and uses the corresponding change in thermopile voltage and the current local temperature to determine the object temperature.

Some of the main characteristics are: response to broad infrared spectrum, no source of bias voltage or current needed and are inherently stable response to DC radiation.

Since the thermopile response is not wavelength dependent, the sensor is built with specific filters for each different application, and its spectral response characteristics are determined only by the transmittance of the window material that act as filter for undesired wavelengths. This is illustrated in figure 4.16.

For instance, the $[5; 14]$ μm region is used for human body detection and the $[2; 5]$ μm region to CO_2 detection.

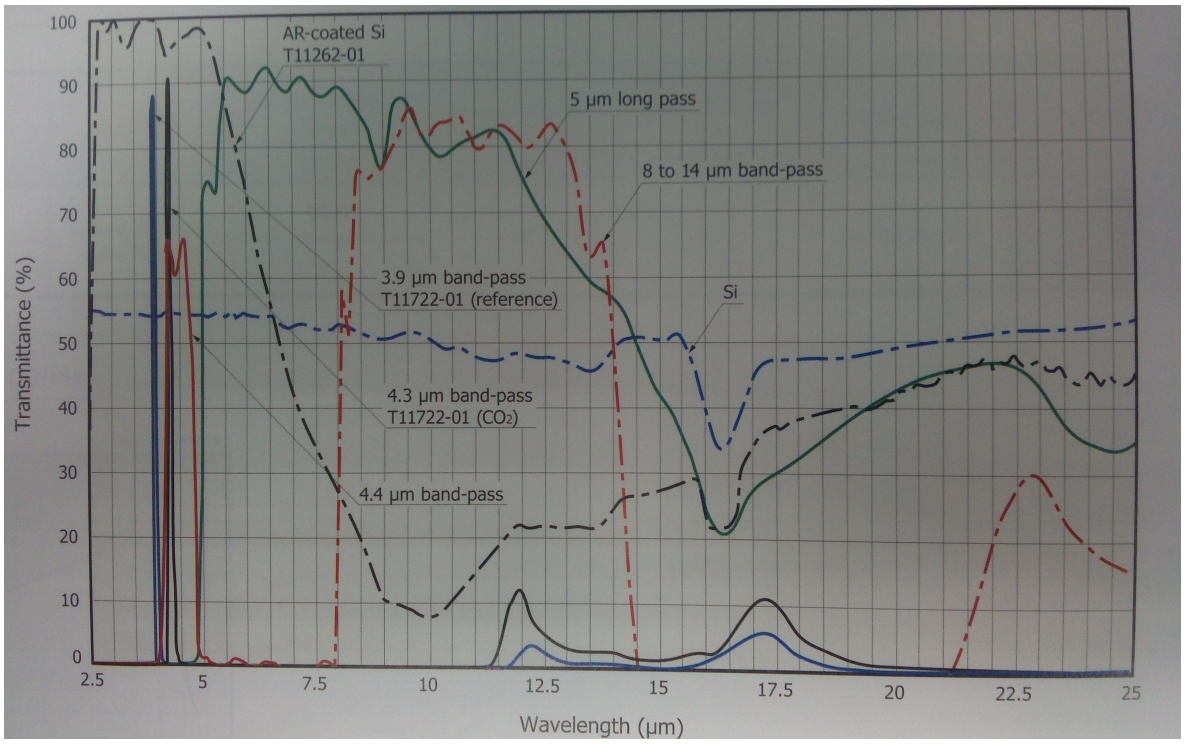


Figure 4.16: Transmittance characteristics of typical window materials. src: HAMAMATSU Selection guide - Infrared Detectors, march 2013.

Figure 4.17 shows the working principle of a thermopile sensor[37]. By specification the working temperature range is from $-40\ ^oC$ to $+125\ ^oC$. According to section 4.1 and (4.2) the working wavelength range is $[7.3; 12.4]\ \mu m$.

The most severe problems arise from temperature swings of the optics. If the sensor cap or the filter lens change their temperature much faster than the sensor base plate, i.e. the sensor itself, then this will act as an additional signal source and influence the sensor output signal. If the heating is large, it can even happen that the amplifier goes into saturation.

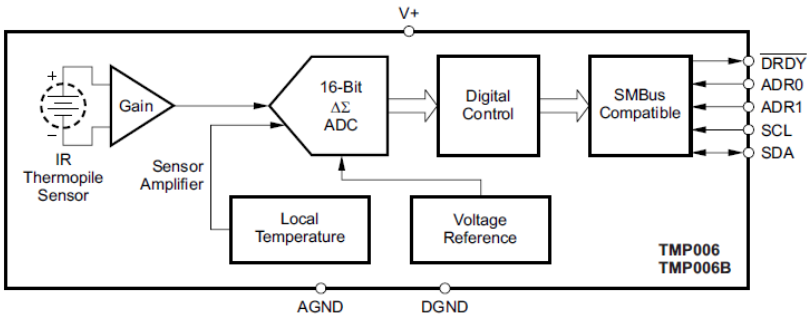


Figure 4.17: TMP006 Block Diagram.

Similarly to the passive infrared sensors, thermopiles have a technical drawback that prevents them from being used for continuous measurements in environments with fast ambient air temperature changes. Since they rely on a thermal detector, as explained in section 4.3,

when measuring, the sensor will heat up, limiting its use to small periods at a time.

4.6 Evaluation of Hamamatsu T11264-08 Dev Module

As a side test, a development board from Hamamatsu (T11264-08(X) [38]) containing a thermopile sensor was tested. It is an 8 x 8 element area array sensor with a preamplifier built-in. It is non-cooled, meaning that in continuous operation the temperature of the thermopile sensor will increase. This is expected, as explained in section 4.3, due to the detector heating up by absorbing incident radiation.

Figures 4.18 and 4.19 present the external appearance of the development board and its block diagram. Figure 4.20 shows the calibration parameters used by default and the control interface.

This experiment consisted in leaving the sensor for some time pointing up towards the ceiling with a distance from it of 3 m in a artificially illuminated room. The objective in this case was not to verify the correct measurement of a object's temperature but to analyze the sensor's output response type and values distribution.

For this trial, 60160 values were obtained. The values in the histogram (4.21) are direct samples from the sensor. No averages were calculated prior to this analysis. These values are fitted by a normal distribution with 40992 values (68.14 %) in the $\mu \pm \sigma$ range, and 57409 (95.43 %) in the $\mu \pm 2\sigma$ range. Normal distribution theory state these values should be 68.95 % and 99.7 % respectively. This discrepancy can be due to the number of samples taken. If they were increased the experimental values would tend to the theoretical ones (assuming normal distribution).

Table 4.4: Normal distribution data.

μ (Mean)	24.5572 °C	σ^2 (Variance)	23.3874

Table 4.5: Development board main components (figure 4.18).

1	Infrared lens.
2	Lens mount.
3	Driver circuit chip.

Another experiment consisted in the detection of a human hand, with an emissivity (η) of 0.99, at a distance of 20 cm.

Since the sensor consists of an array of 64 cells, figure 4.22a shows the raw data of 2 different cells in two distinct positions within the array (matrix). This response was expected due to the previous experiment. In order to mitigate this behavior, an operation of moving average can be performed (4.22b). This allows to remove high value discrepancies between samples and achieve a much smoother signal.

Although, when considering the average value of the entire raw data matrix (figure 4.22c), for each sample, is possible to detect the pattern of the movement in front of the sensor.

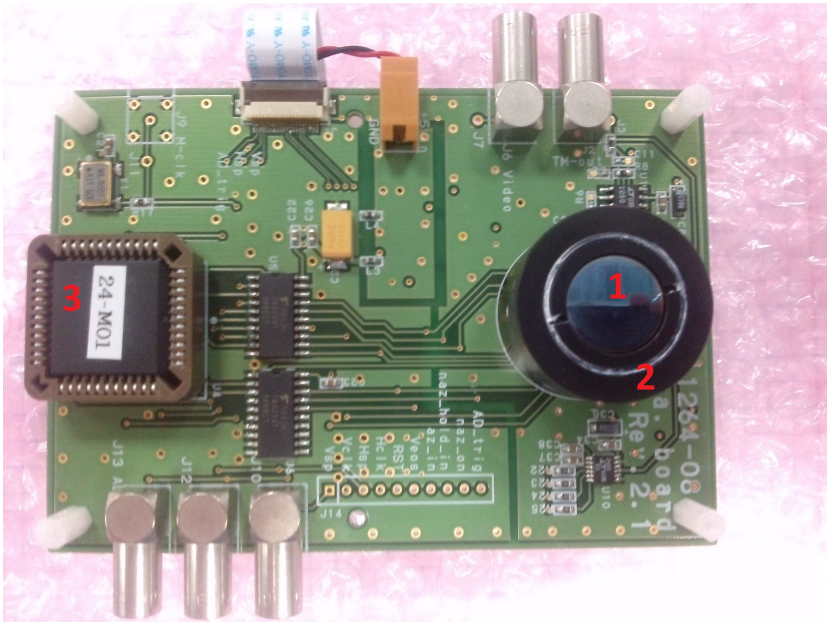


Figure 4.18: Hamamatsu T11264-08(X) Development Board.

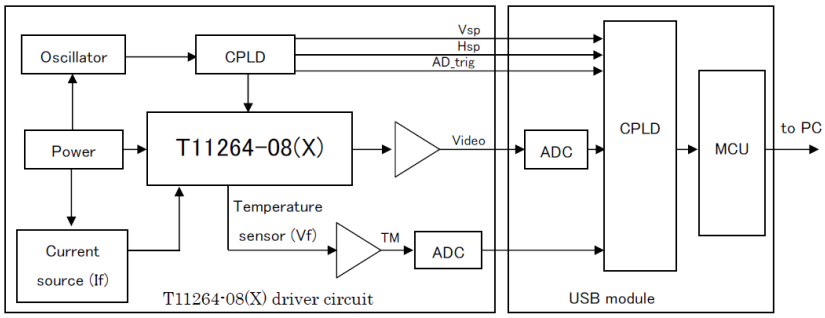


Figure 4.19: Hamamatsu development board block diagram.

Performing a moving average on the average signal of the entire matrix, a much smoother one is obtained. Figure 4.22d. After this, another test was performed by changing the object to be tested to a cardboard with an emissivity of 0.81, lowering the distance to 7 cm. The results are shown in figure 4.23.

An evaluation of the Hamamatsu board was made to highlight the major concerns presented earlier in this chapter. This allowed to verify the noise presence (mostly white), even when the system is not stimulated. Finally, a method to smooth the output signal in order to use the thermopile sensor as presence detector was tested.

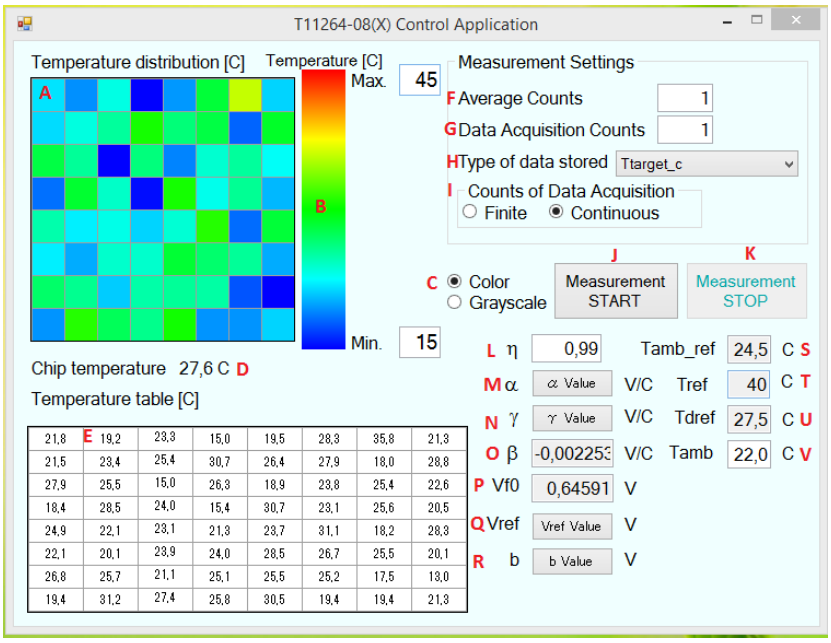


Figure 4.20: Software User Interface.

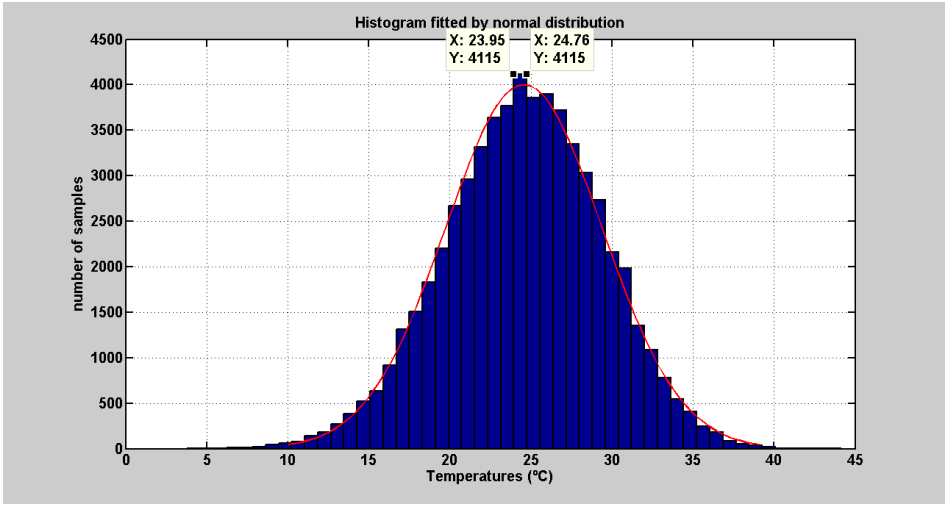


Figure 4.21: Measured values fitted by a normal distribution.

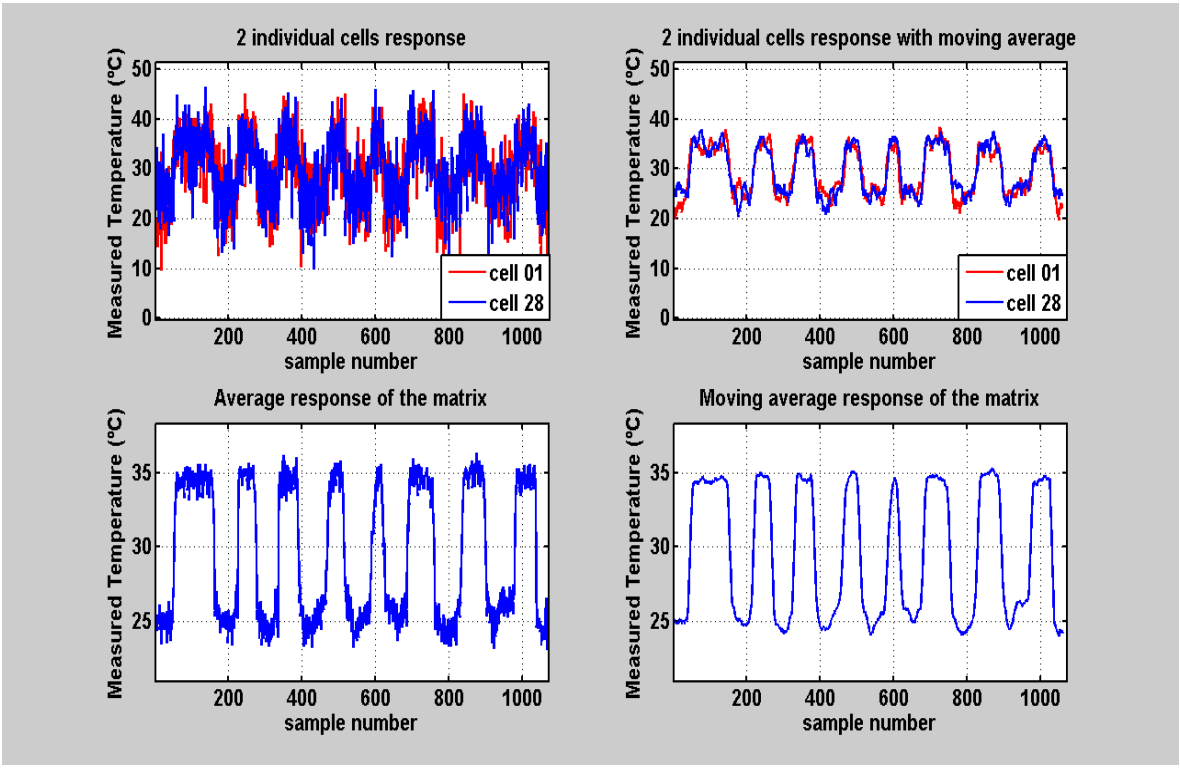


Figure 4.22: Sensor response: a. 2 different cells; b. 2 different cells with moving average; c. matrix average response; d. moving average of c. - Moving averages with 12 samples.

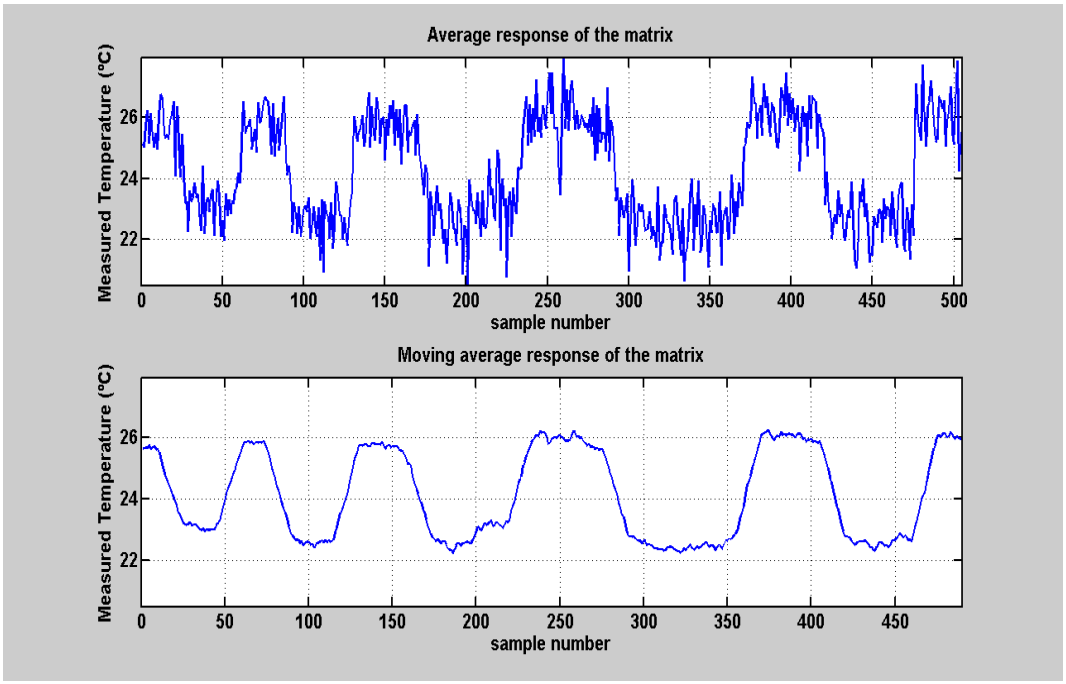


Figure 4.23: Average response for a cardboard object ($d \approx 7$ cm).

4.7 Chapter Remarks

This chapter started with a brief theoretical introduction to infrared radiation which highlighted the main concerns to take into account when trying to detect this radiation. Some characteristics of existing detectors were presented and two active sensors were tested. These tests included a spectral analysis to a PWM signal. This allowed to demonstrate the importance of the frequency contents in a signal used for presence detection.

Thermopile sensors' main characteristics were also presented. As an example of how these sensors work, a board from Hamamatsu was tested.

Chapter 5

Project Implementation

The previous chapters allowed for an understanding of the main issues when designing and implementing a low-power system with energy harvesting. It was also shown the main challenges when dealing with infrared detection.

In this chapter, an implementation with all three previous components is designed and tested. The system consists of a low-power, "smart" faucet with infrared detection and energy harvesting. The chapter will start by describing the system and the main issues to be addressed. After this, the project design choices will be presented, primarily focusing the hardware. At this time, a theoretical energy balance analysis will be performed that allows to achieve a model for the system's operation. On the second part, after the assembly of the system's hardware, several software optimizations to improve energy and water efficiency are proposed. Finally, the results in terms of power consumption, energy harvesting and reliability of the infrared sensor are presented and a conclusion for each topic is drawn.

5.1 System's Description and Design Considerations

As stated above, the main goal is to design a "smart" faucet with infrared detection and energy harvesting. Common automatic faucets contain only a valve and an infrared sensor and are either AC or battery powered.

The new system will be based on the common automatic faucets with some "intelligence" added to it. It will be based on a microcontroller, a bistable valve, an infrared sensor and, for energy harvesting, an hydro-generator.

Starting with the microcontroller, it must be specifically designed for low-power applications, meaning that there must be hardware and software tools that allow power optimization at all levels.

The valve is chosen to be bistable due to the fact that it only needs a small trigger to transition from closed/open to open/closed state. This makes the current consumption of the valve independent of the time it remains open or closed.

Concerning the infrared detection, the sensor should be as reliable as possible while consuming very little power.

Finally, the hydro-generator must be adequate to the application at hand. A small water flow should be enough to trigger the power generation. Furthermore, the use of an energy harvester will be carefully measured, in an attempt to make the system self-sustainable.

A very important issue to take into account is the necessary technology to assemble the

entire system. Several top-of-the-line components with extreme energy efficiency are available, but are not usable in this prototyping context. All the components need to be hand mounted, and therefore component packaging is an eliminatory property.

On the software part, low-power modes should be used whenever possible and unnecessary hardware should be disabled. This should also include shutting down the infrared sensor allowing major power savings. It is necessary to take into account that, when the entire system enters in a low-power mode, no detection is performed, making the entire system useless in the user's perspective. Is then necessary to wake up within acceptable periods. This still allows for major power savings.

An important software issue is the need to increase the reliability of the infrared sensor. It is necessary to implement a light (and fast) algorithm to perform this task.

5.2 Hardware Interfaces and System Assembly

In order to start choosing the main system components, is mandatory to verify the already available tools. This is particularly relevant when choosing the microcontroller's technology, packaging and manufacturer. At this point, with the information provided by chapter 2 the microcontroller to be chosen would be the MSP430FR family. The necessary tools to develop in this platform needed to be bought which would increase the project costs. It is then necessary to choose a microcontroller whose tools are already available - the ATmega48 from ATMEL. Other integrated circuits' packaging also needed to be carefully chosen due to the prototyping process: all components were assembled by hand.

Table 5.1: ATmega48 electrical main characteristics.

Supply Voltage	[2.7 ; 5.5] V
V_{OH}	4.2 V @ $V_{DD} = 5$ V
V_{OL}	0.7 V @ $V_{DD} = 5$ V
V_{IH}	$0.6V_{DD}$ V @ $V_{DD} = 5$ V
V_{IL}	$0.3V_{DD}$ V @ $V_{DD} = 5$ V

For the infrared sensor, by the analysis performed in subsection 4.5.2, the appropriate choice is the TSAL6100 & TSSP58P38 combination. Low power consumption, high versatility and reliability are the key aspects of this sensor. Although the SHARP 2Y0A21 F 9Y sensor is more reliable, it needs to be connected to the ADC due to the non-linearity of its output. This would increase the current consumption of the microcontroller.

Table 5.2: TSAL6100 & TSSP58P38 electrical main characteristics

TSSP58P38: Supply Voltage	[2.5 ; 5.5] V
TSSP58P38: Supply Current	0.9 mA
TSAL6100: Forward Voltage	1.6 V

Table 5.3: EHCOTECH DDT-ML-4.5 VDC electrical main characteristics

Supply Voltage	[4.5 ; 9] V
Supply Current	500 mA @ 5 V
Pulse Width	50 ms

Table 5.4: Hydro-generator F50-5V 10W main characteristics

Output Voltage	5 V
Output Power	10 W
Start Pressure	0.5 bar

As stated above, the valve to be selected should be bistable in order to save power. Mainly due to its low price and availability a valve from EHCOTECH was chosen.

The use of micro hydro-generators isn't a common practice yet. Therefore in order to buy a generator in such a small scale is necessary to go to ebay. There, an hydro-generator F50-5V 10W was found.

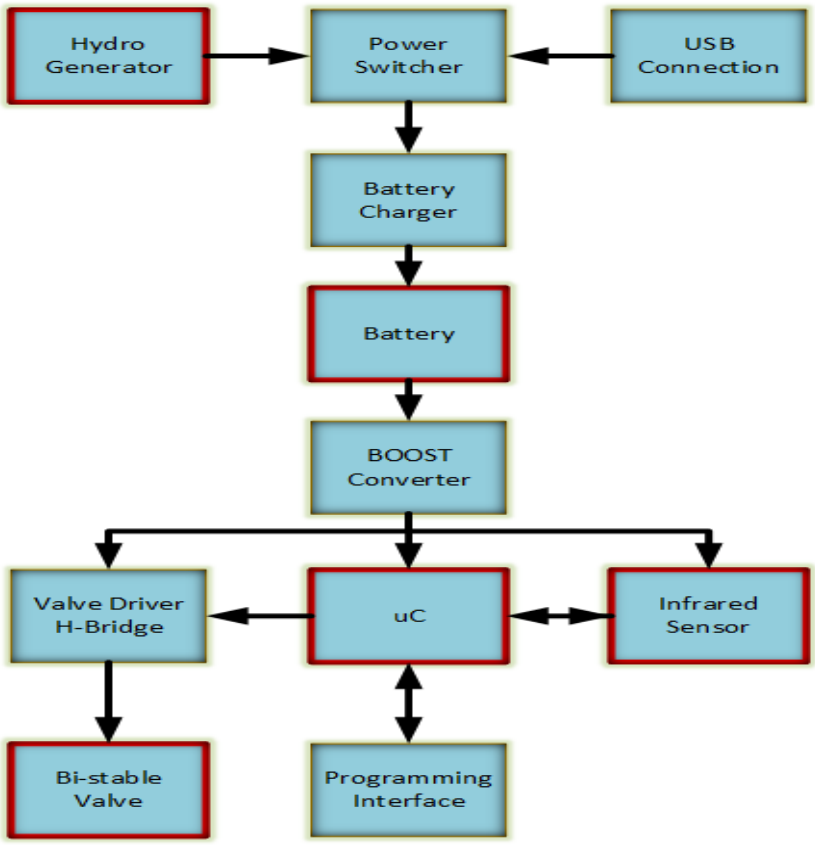


Figure 5.1: Block diagram of the faucet system.

After selecting the main components of the system is necessary to ensure they are compatible with each other and make the necessary adjustments. Figure 5.1 shows the main components and interfaces.

The hydro-generator and an USB port (the use of USB allows for an easy battery charge with widely available chargers) are connected to a power switcher. This allows to switch between the two sources of power. The preferred power source is the hydro-generator, meaning that if both are providing energy, the power switcher disconnects the USB.

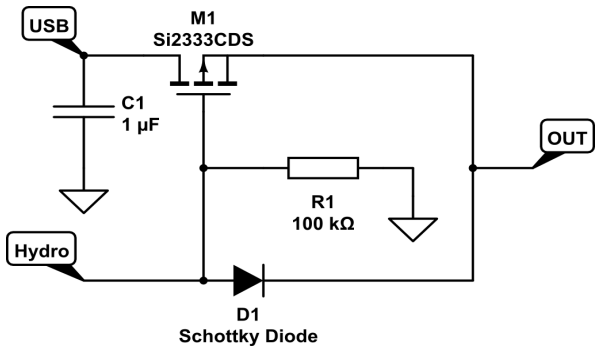


Figure 5.2: Power Switcher Circuit

The next interface circuit is between the power switcher and the battery, since the Li-Ion/Li-Polymer batteries need for a constant-current/constant-voltage algorithm provided by the battery charger. Furthermore, it should allow for a programmable charge current.

The interface between the battery and the remaining circuitry is made with a boost converter. This is necessary due to the battery discharge process. With a boost converter the remaining circuitry becomes rather independent on the battery voltage as long as it contains a minimum amount of charge.

Finally, since the valve needs some significant power to switch between states, it can't be connected directly to the microcontroller. A H-bridge allows for a power switching between the logic levels of the microcontroller and the necessary power levels.

Table 5.5: Components used.

Microcontroller	ATmega48-20AU
Battery	LP603048 3.7 V 570 mAh
Infrared sensor	TSAL6100 & TSSP58P38
Valve	EHCOTECH DDT-ML-4.5 VDC
Hydro-generator	Water generator F50-5V 10W
Battery Charger	MCP73831T-2ACI/OT
Boost Converter	MCP1642B-ADJI/MS
H-Bridge	SiP2100
2×LED	OSRAM 1206

Figure 5.4 shows the assembled PCB.

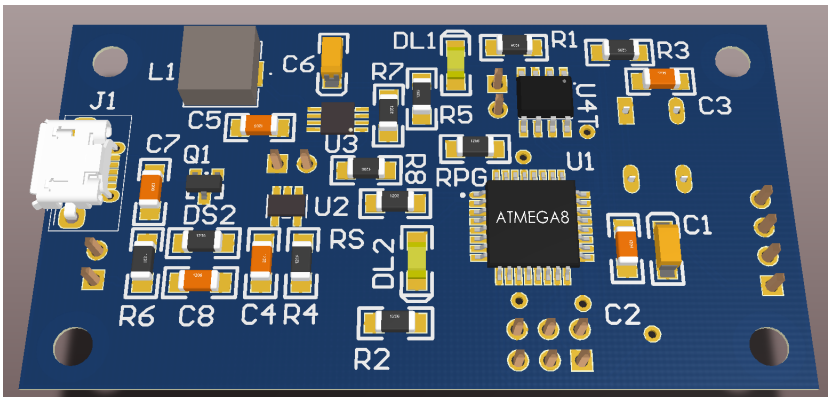


Figure 5.3: Top view of the PCB.

5.3 Energy Balance

After the design of the entire system, is necessary to perform an energy balance to assess its self-sustainability. The first step is to draw a temporal diagram to increase the perception in how the system works.

The microcontroller and the infrared sensors work with a duty-cycle to save power. The time between sensor activations for presence checks must acceptable for the end user, but also needs to be big enough to save significant amounts of power.

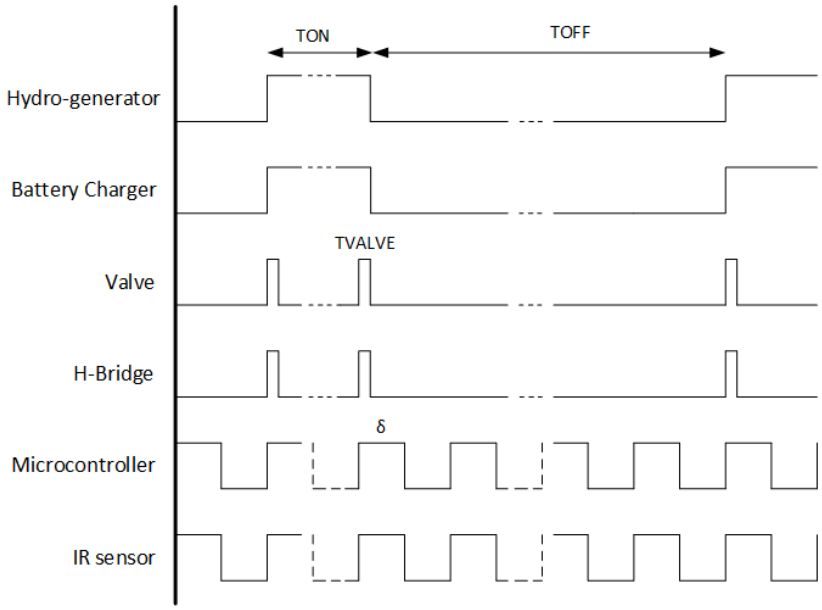


Figure 5.4: Timing diagram

As explained earlier, the valve is bistable to allow for major power savings by making its power consumption independent from the time it remains on or off. The constant t_{Valve} is the amount of time the valve needs to switch states (0.05 seconds in this case) and the parameter δ is defined as the percentage of time the microcontroller and the infrared sensor are active in a second: $\delta \in \ ]0;1]$. t_{ON} is the amount of time the water is running and t_{OFF} is the time until

next activation. The equations derived from the timing diagram, in order of appearance, are:

$$Q_{\mathrm{HG}} = t_{\mathrm{OFF}} \cdot i_{\mathrm{HG\ min}} + t_{\mathrm{ON}} \cdot i_{\mathrm{HG\ max}} \tag{5.1}$$
$$Q_{\mathrm{BC}} = t_{\mathrm{OFF}} \cdot i_{\mathrm{BC\ min}} + t_{\mathrm{ON}} \cdot i_{\mathrm{BC\ max}} \tag{5.2}$$
$$Q_{\mathrm{VV}} = (t_{\mathrm{OFF}} - 2t_{\mathrm{Valve}}) \cdot i_{\mathrm{VV\ min}} + 2t_{\mathrm{Valve}} \cdot i_{\mathrm{VV\ max}} \tag{5.3}$$
$$Q_{\mathrm{HB}} = (t_{\mathrm{OFF}} - 2t_{\mathrm{Valve}}) \cdot i_{\mathrm{HB\ min}} + 2t_{\mathrm{Valve}} \cdot i_{\mathrm{HB\ max}} \tag{5.4}$$
$$Q_{\mathrm{uC}} = (t_{\mathrm{OFF}} + t_{\mathrm{ON}})(1-\delta) \cdot i_{\mathrm{uC\ min}} + (t_{\mathrm{OFF}} + t_{\mathrm{ON}})\delta \cdot i_{\mathrm{uC\ max}} \tag{5.5}$$
$$Q_{\mathrm{IR}} = (t_{\mathrm{OFF}} + t_{\mathrm{ON}})(1-\delta) \cdot i_{\mathrm{IR\ min}} + (t_{\mathrm{OFF}} + t_{\mathrm{ON}})\delta \cdot i_{\mathrm{IR\ max}} \tag{5.6}$$

Giving the 'plus' signal to the amount of energy provided to the system, and the 'minus' to the energy consumed, the total amount transfered is:

$$Q_{\mathrm{T}} = +Q_{\mathrm{HG}} - Q_{\mathrm{BC}} - Q_{\mathrm{VV}} - Q_{\mathrm{HB}} - Q_{\mathrm{uC}} - Q_{\mathrm{IR}} \quad [\mathrm{C}] \tag{5.7}$$

Considering each individual maximum and minimum currents for each component as constants, the energy balance will be a function of t_{ON}, t_{OFF} and δ: $Q = f(t_{\mathrm{ON}}, t_{\mathrm{OFF}}, \delta)$. For this particular case, with the values in table 5.6, where the 1.04 factor corresponds to the 96 % efficiency off the boost converter, the energy balance function is:

$$Q(t_{\mathrm{ON}}, t_{\mathrm{OFF}}, \delta) = 0.600t_{\mathrm{ON}} - 0.020\delta(t_{\mathrm{ON}} + t_{\mathrm{OFF}}) - 0.002t_{\mathrm{OFF}} - 0.052 \quad [\mathrm{C}] \tag{5.8}$$

Solving for t_{ON}, which gives the amount of time the valve needs to be open in order to generate enough power to match the energy spent in T_{OFF}:

$$t_{\mathrm{ON}} = \frac{t_{\mathrm{OFF}} + 10\delta t_{\mathrm{OFF}} + 26}{300 - 10\delta} \quad [\mathrm{s}] \tag{5.9}$$

Table 5.6: Current values for each component.

Device	i_{min} [A]	i_{max} [A]
Hydro-generator	0	$600 \cdot 10^{-3}$
Battery Charger	$2 \cdot 10^{-6}$	$510 \cdot 10^{-6}$
Valve	0	$1.04 \times 500 \cdot 10^{-3}$
H-Bridge	$1.04 \times 50 \cdot 10^{-6}$	$1.04 \times 200 \cdot 10^{-6}$
Microcontroller	$1.04 \times 540 \cdot 10^{-6}$	$1.04 \times 1 \cdot 10^{-3}$
Infrared sensor	$1.04 \times 900 \cdot 10^{-6}$	$1.04 \times 20 \cdot 10^{-3}$
LED	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$

This analysis clearly shows that the major power consumption contributors are the microcontroller and the infrared sensor due to the periodic activations. Figure 5.5 shows the necessary t_{ON} to achieve energy balance zero as a function of t_{OFF} for $\delta = 0.5$.

The minimum t_{ON} for the open/close operation to be self-sustained can be calculated by making $t_{\mathrm{OFF}} = 0$ and $\delta = 1$ in (5.9), resulting in a $t_{\mathrm{ON\ min}} = 0.09$ seconds.

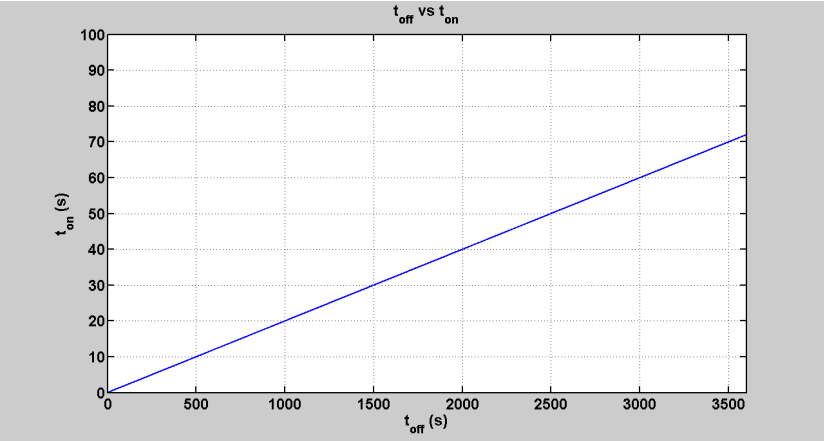


Figure 5.5: Necessary t_{ON} to generate enough energy to overcome t_{OFF}.

5.3.1 Case Study

To evaluate the applicability of this model for a real situation, a simulation can be performed in a restaurant's restroom in which the faucet is installed. The timing characterization of this particular case is described by two intervals with high use followed by sporadic or no uses at all. Figure 5.6 shows the described situation.

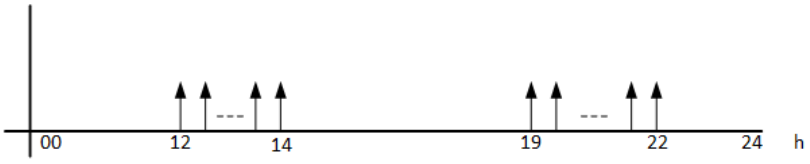


Figure 5.6: Faucet use timing diagram.

Starting with the always off intervals, [00 ; 12] h, [14 ; 19] h and [22 ; 24] h and using (5.8) with $\delta = 0.5$, the total energy loss is:

$$\begin{aligned} t_{\mathrm{OFF}} = (12 - 00) \cdot 3600 &\Longrightarrow Q_T = -507.8 \text{ C} \\ t_{\mathrm{OFF}} = (19 - 14) \cdot 3600 &\Longrightarrow Q_T = -211.6 \text{ C} \\ t_{\mathrm{OFF}} = (24 - 22) \cdot 3600 &\Longrightarrow Q_T = -084.7 \text{ C} \\ &\qquad\qquad = -804.1 \text{ C} \end{aligned}$$

If the restaurant has 20 tables, with an average of 2 persons per table and 2 services per time interval, this gives 80 persons that can use the faucet in a given period. Considering that a single use lasts for about 10 seconds, in the [12 ; 14] h time interval, the average time between uses, t_{OFF}, in seconds is:

$$t_{\mathrm{OFF}} = \frac{(14 - 12) \cdot 3600 - 80 \cdot 10}{80} = 80 \text{ s} \tag{5.10}$$

Replicating this equation for the [19 ; 22] h period, t_{OFF} equals 125 s. According to (5.9), the necessary t_{ON} to make the system self sustainable for the calculated t_{OFF} is 1.72 and

2.63 seconds respectively, meaning the system is accumulating energy when considering the average use of 10 seconds. The total energy accumulated in each period is given by (5.8):

$$\begin{aligned} 80 \cdot Q(10, 80, 0.5) = 80 \cdot 4.89 &= +393.2 \text{ C} \\ 80 \cdot Q(10, 125, 0.5) = 80 \cdot 4.35 &= +350.9 \text{ C} \\ &= +744.1 \text{ C} \end{aligned}$$

The final energy balance is then -60.0 C, meaning that in these conditions the system is not self sustainable. In this model, when the only programmable and changeable parameter is δ, is necessary to carefully find a balance between optimal sensor/faucet's performance and energy consumption. An appropriate change in this parameter may bring the system for a positive energy balance. In this case, $\delta = 0.46$ is sufficient.

When performing this analysis, there are some key aspects that need to be taken into account. For instance, after the biggest t_{OFF} period, [22 - 12] h, there must be enough energy stored in the battery for the system to start. This means that, in the high use periods, the battery must charge enough to overcome the off periods with sufficient energy for the system to start again. Another issue that rises is the fact that battery storage capacity is limited. Even if there is more than enough energy available to charge the battery in high use periods, it will only charge up to its limit. Therefore, energy for off periods may still not be enough, meaning that battery capacity must not be overlooked. In short, battery capacity determines the maximum t_{OFF}.

5.4 System Programming

In order to include the previously proposed optimization model into the microcontroller, is necessary to start by drawing a flow diagram. This diagram includes the main power states and major program decisions and sequences.

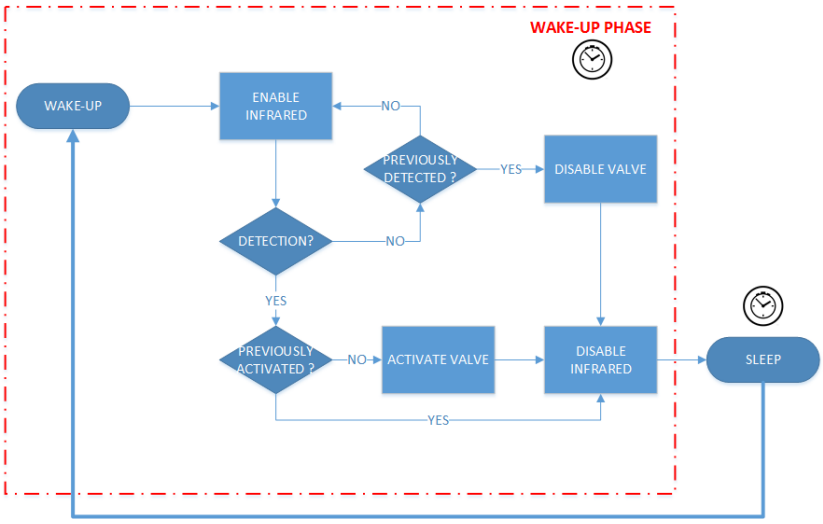


Figure 5.7: Program flowchart.

The clock symbol illustrates the timed events reliant on δ. The "ENABLE INFRARED" $\rightarrow$ "DETECTION?" $\rightarrow$ "PREVIOUSLY DETECTED ?" loop is kept active only while the system is awake. When a sleep trigger is received, the system enters in the sleep mode.

From the flowchart it is possible to introduce a new optimization: the time spent on the wake-up phase also depends on the interactions that occur and can be significantly less than δ, as long as a detection is made in the beginning of the phase. It can also be extended for a short period if a detection is made at the end of the wake-up phase. In this case, the system must be awake long enough to complete the remaining phases until the sleep phase. Nevertheless, shutting down the infrared sensor and going to sleep as soon as a detection is made can save a lot of energy.

From the timing diagram and from the flowchart, in the figures 5.4 and 5.7 respectively, several signal frequencies can be identified: the generation of the PWM signal, the wake-up/sleep cycle and the time the valve needs to turn on/off.

As already introduced in section 4.5.2.1, a 38 kHz square signal is necessary to perform the detection. It was also shown that a 50 % duty cycle was more energy efficient.

Since the ATmega48 microcontroller possesses only a clock source shared by the CPU and the peripherals, it is necessary to find a way to have a clock generator compatible with all the time constants. The most problematic frequencies are the 38 kHz and $1/(1-\delta)$ Hz, due to the magnitude difference. To achieve this compatibility, is necessary do decrease the clock frequency to 500 kHz. This ensures the minimum achievable frequency of 1 Hz as well as the 38 kHz PWM signal. This is based on the mathematical expressions for PWM on Timer1 and clear timer on compare match (CTC) on Timer2:

$$f_{\text{OC1A PWM}} = \frac{f_{\text{clk I/O}}}{N(1 + \text{OCR1A})} \tag{5.11}$$

$$f_{\text{OC2A CTC}} = \frac{f_{\text{clk I/O}}}{2N(1 + \text{OCR2A})} \tag{5.12}$$

$$\text{N (prescale factor)} = 1,\ 8,\ 64,\ 256,\ \text{or}\ 1024$$

This yields OCR1A = 12 (38.4 kHz) and OCR0A = 243 ($\approx$ 1 Hz), with N = 1 and N = 1024. This clock frequency reduction has a direct impact on the available frequency resolution for the 38 kHz signal. The direct consequence is a small loss in the infrared receiver, since its peak sensitivity is at 38 kHz.

This is the most practical way of generating these signals. On Timers 0 and 2 the maximum allowable frequency is 31 kHz, and the only timer with the ability to wake-up the microcontroller is the Timer2. The other option, without using the PWM module is to make an interrupt service routine to toggle an output bit at the rate of 38 kHz. This method creates a gigantic overhead on the microcontroller. Furthermore, the microcontroller would be almost always servicing the interrupt routine, making the remaining code execution very slow.

The next step is to increase the reliability of the infrared sensor whose output is active low when a 38 kHz signal is detected. This is made by a debouncing algorithm shown in figure 5.8. This is implemented in the "DETECTION?" block in the program flowchart (figure 5.7).

The clock symbol represents a timer triggered event. In the timer's interrupt service routine, the sensor pin will be read and a variable will be incremented or decremented depending on the presence of an object in front of the sensor. When the variable reaches its maximum or minimum value, the presence (or absence) of an object is validated. The time until validation must be small enough due to the "duty cycle" nature of the system (recall the timing diagram

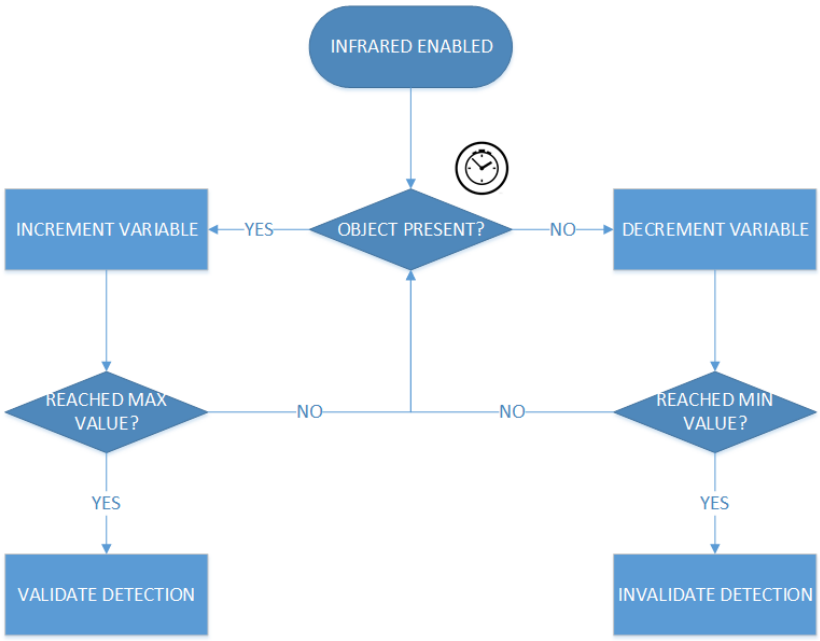


Figure 5.8: Debouncer flowchart.

on 5.4). The time spent sleeping added to the time until a positive validation is generated (or faucet activation) must be made acceptable by the end user.

The verification is made using Timer0. The chosen maximum time to increment a variable up to 255 for a successful validation was 100 ms. The necessary frequency is in the range of 2.5 kHz. As explained earlier, by disabling the infrared detection and debouncing corresponding timers, when a positive validation is performed, allows major power savings.

The next time constant to take into account is the valve activation time. It needs 50 ms to change between states. There are a few considerations to take into account. First, in order to save power, the valve must be actuated only during the necessary amount of time. After this, its inputs must be both deactivated. Since it is the last step before entering sleep, is it worth to reconfigure a timer to use only once and then change it to its default values? The necessary steps would be: 1. deactivate all timers; 2. open the valve; 3. configure Timer0 to generate an interrupt; 4. go to sleep; 5. wake-up 50 ms after to deactivate the signal applied to the valve; 6. reconfigure Timer0 to the previous state; 7. go to sleep the remaining time. Finally, it would be necessary a global control variable for the interrupt service routine, since it was already configured with the debouncing algorithm. It is important to keep in mind that the system clock is configured at a quite low speed (500 kHz). The overhead time of changing contexts and the wake-up/sleep process, plus the previous steps, would be too long. Alternatively, the *_delay_ms*(50) function can be used to perform this operation, which keeps the CPU trapped until the end of the necessary time.

This open loop control algorithm works properly if the system behaves seamlessly. To protect against miss-executions, the program must use the STAT pin from the battery charger, which indicates if the battery is charging or not. This can be used as a confirmation if the valve was effectively open or closed. Furthermore, in order to know the state of the battery, the PGOOD pin from the boost converter must be read. This allows to warn the system administrator that the battery is running low. By using both status pins a more reliable

closed loop control algorithm can be implemented.

According to the mathematical model presented earlier, the only variable that is not user dependent is δ. This parameter can be changed dynamically according to the faucet use. That is, the program can adjust δ if t_{ON} is not long enough to overcome t_{OFF}. Since it will make the system's response slower it is necessary to impose reasonable limits to δ.

The used code is shown in annex C.

5.4.1 Detected Issues

Using an oscilloscope to verify the generated signals (PWM 38.4 kHz, 2.5 kHz and $^{1}/_{(1-\delta)}$ Hz), was verified that the PWM signal had a frequency of 19.2 kHz. This revealed an error in (5.11) in the datasheet which omitted a division by two factor. The fastest approach was to make OCR1A = 6. The immediate implication was the loss of precision in the frequency generation. From (5.11) with the division by two correction, the achieved frequency was 35.7 kHz, or with OCR1A = 5, 41.7 kHz. The analysis of figure 5.9 [35] shows the energy loss when using both frequencies.

To achieve the same detection distance is necessary to increase the emitter current by at least 40%.

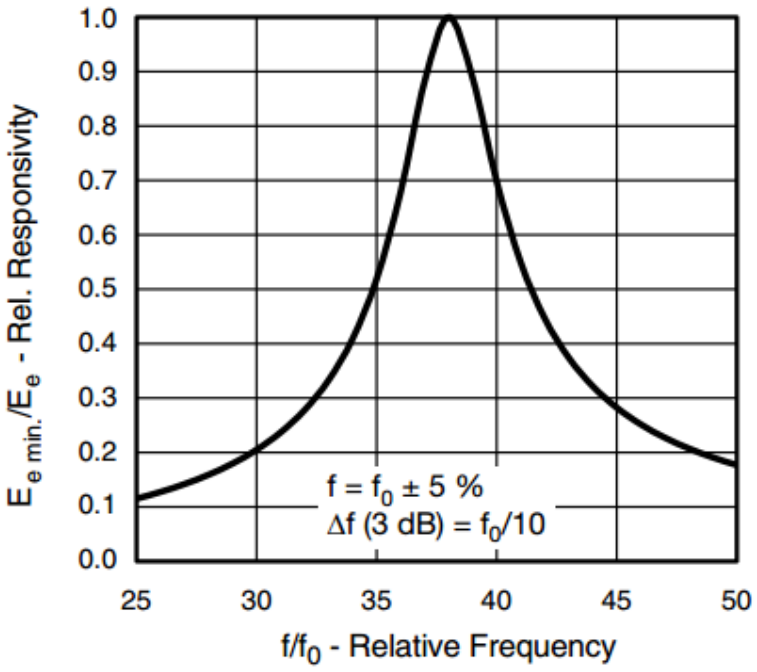


Figure 5.9: Frequency Dependence of Responsivity

5.5 Experimental Results

In order to assess the system's performance in terms of power consumption and energy harvesting, is necessary to use a precision ammeter. Some different objects were tested to

assess the detection distance of the infrared sensor.

Table 5.7 shows the obtained current values for each major system. For the combination PCB + IR sensor the minimum current consumption is mainly due to the always on LED (6 mA). Also during t_{ON}, another debug LED was enabled when a detection was triggered. This increased the current consumption of the system for another 6 mA. With several consecutive measurements, was noticeable that the valve's current consumption was only around 150 mA, instead of 450 mA as initially estimated.

The hydro-generator needed a more complex analysis. With the battery initially charged, was possible to measure a 32 mA current generation (with a water pressure of around 3 bar). As the battery started to discharge, the generated current increased significantly, reaching 283 mA when the battery presented 3.5 V at its terminals. These variations can be explained by the algorithm implemented by the battery charger (constant-current / constant-voltage). The 283 mA current was imposed by the charge resistor to increase battery life. A 3k3 Ohm resistor was placed to limit the charge current to 300 mA. With only a resistive load applied to the hydro-generator's terminals, the current generated was 475 mA, with a water pressure of 3 bar.

Table 5.7: Measured Currents

Parts	i_{min} [mA]	i_{max} [mA]
PCB + IR sensor	-6.2 ±0.05	-36 ±0.05
PCB + IR sensor + Valve	-36 ±0.05	-188 ±0.05
Hydro-generator	0 ±0.05	[32 ; 283] ±0.05

Table 5.8: Infrared sensor distances

Object	Distance [cm]
Hand	24 + 2 ± 0.5
Black Plastic	7 + 1 ± 0.5
White Foam	27 + 2 ± 0.5

In these experiments, the infrared sensor showed some hysteresis when activating and deactivating the system. For instance, when placing a hand to activate the sensor, the maximum distance was 24 cm. To deactivate it, was necessary 26 cm.

The necessary calculations, for the obtained experimental results, can be derived from (5.1)-(5.6):

$$\begin{aligned} Q_{\mathrm{T}} = &- 6.2 \cdot 10^{-3}(t_{\mathrm{OFF}} + t_{\mathrm{ON}})(1-\delta) \\ &- 36 \cdot 10^{-3}(t_{\mathrm{OFF}} + t_{\mathrm{ON}})\delta \\ &- (188-36) \cdot 10^{-3} t_{\mathrm{Valve}} \\ &- 6 \cdot 10^{-3} t_{\mathrm{ON}} + 175 \cdot 10^{-3} t_{\mathrm{ON}} \end{aligned}$$

$$t_{\mathrm{ON}} = \frac{211 t_{\mathrm{OFF}} - 31\delta t_{\mathrm{OFF}} + 76}{31\delta + 634} \quad [\mathrm{s}] \tag{5.13}$$

Figure 5.10 represents (5.13) for $\delta = 0.5$. Figure 5.11 shows the results in which the current consumption of the LEDS was removed.

Recalling figure 5.5, a great increase in the necessary t_{ON} to overcome t_{OFF} is noticeable. For instance, for $t_{\mathrm{OFF}} = 500$ s the increase is from $t_{\mathrm{ON}} = 10$ to 58.4 seconds. The major differences from the theoretical model to the experimental results lie in the infrared sensor current increase as explained earlier and the reduced generated current from the hydro-generator.

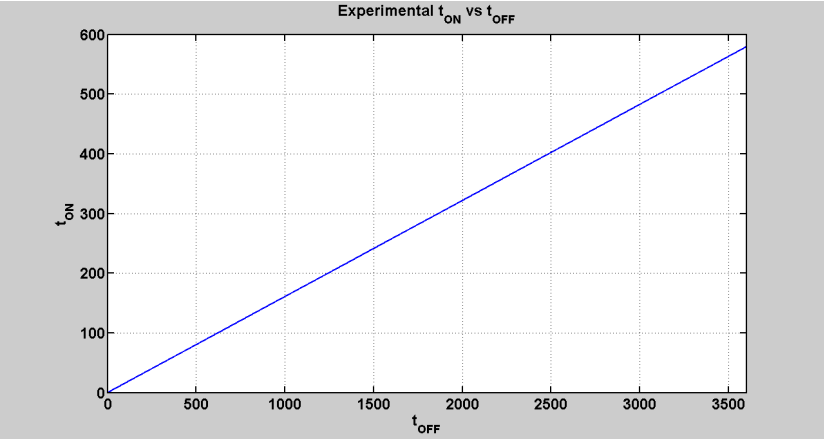


Figure 5.10: Experimental results for $\delta = 0.5$.

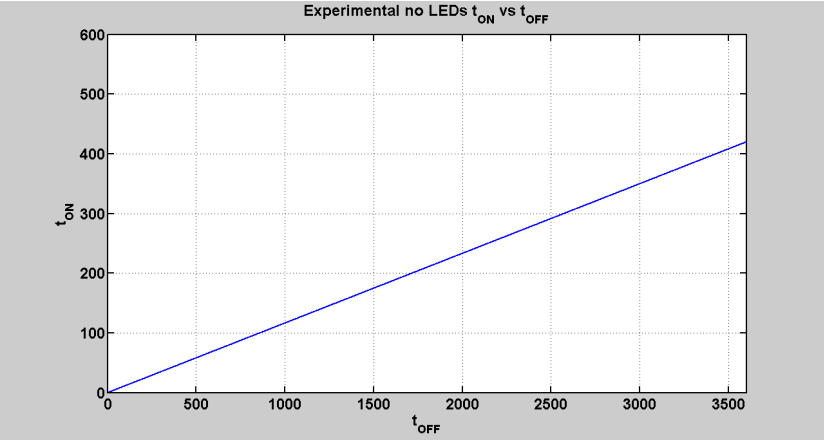


Figure 5.11: Experimental results for $\delta = 0.5$ with the LEDs removed.

5.6 Chapter Remarks

This chapter started by presenting the major design considerations when implementing a low-power system with energy harvesting and infrared detection. After this, the available tools were assessed in order to choose the main hardware components. The necessary steps

to design and assemble the test circuit were presented. An energy balance with a case study allowed to estimate the battery life in a specific application.

Finally, the experimental results showed the major limitations when the hardware component choices are limited from the start. The available tools to manufacture the PCB were the major limitation, which forced the use of less efficient components. Then, the microcontroller used was very inefficient for this application. A single clock source, few timer configuration modes and high power consumption were the major constraints of this microcontroller. Lastly, the battery used should have a higher capacity in order to use the full current generation capability of the hydro-generator.

In terms of infrared presence detection, the higher current consumption, due to the necessary configurations in the microcontroller, was an unexpected drawback which affected the expected experimental results.

Chapter 6

Conclusions and Future Work

The main objective of this dissertation was to develop a system with low-power consumption and high reliability in terms of infrared detection. The chosen path, was not only make use of the best practices to develop low-power systems, but also increase battery life using energy harvesting. For this, the end goal was to develop a "smart" faucet system to implement the necessary procedures to achieve the high energy efficiency and infrared reliability.

The procedures to build this system, starting from the hardware up to the software layer followed in this dissertation, had the objective to highlight the key aspects and issues to take into account when designing a system with these main objectives in mind: low-power consumption and highly reliable infrared detection.

In chapter 2, was possible to identify the key aspects and solutions to achieve the lowest power consumption possible. The presented case study allowed to apply several advices, introduced throughout the chapter, and verify its importance. Although most advices can be widely used in most microcontrollers, it is important to know the hardware architecture and software compiler's principles of operation in order to achieve the lowest power consumption.

The energy harvesting methods presented in chapter 3 can be used to increase the system's battery autonomy. Since of these systems provide energy in the range of the μW, the harvesting circuit must be as efficient as possible. Although a simple overview of each method was presented, it gave the knowledge to implement an harvester for the proposed faucet system.

Infrared detection, by itself is a large topic with a wide range of concepts. With the objective of high reliability at hand, the main concepts were presented. After this, by performing an analysis to the most common sensors, was possible to identify the most suitable sensor to use in this application.

When implementing the three previous major concepts in a single system it is important to verify each compatibility angle between each major system and component. This became evident, for instance, when trying to generate very low and very high frequencies with a single clock source. Furthermore, it is important to test each system separately and stimulate it in all necessary situations to avoid miss-calculations. The time spent in these assessment tasks can be compensated when assembling the entire system.

The fake positives/negatives rejection due to the debouncing method was not properly measured due to amount of work and trials necessary to gather enough data to draw a proper conclusion. However, combined with the sleep mode, the amount of time to trigger the faucet was long enough to suppress unwanted activations.

By shutting down every unnecessary peripheral during the sleeping periods, allowed to

achieve a standby current of 200 μA. The energy harvesting system performed quite well even with the imposed limitations. The generated power, although not constant due to the battery charge manager, allowed to prove the applicability of this concept in more efficient systems.

Although positive results have been achieved, there is always margin for improvements and, in this section, a few suggestions are left for possible future work. It would be important to start by testing this system, as is, in a real environment, similar to the theoretical case study presented in chapter 5. As an ultimate proof of concept, a new design with more efficient components should be tested to make a real assessment of the capabilities of the energy harvesting platform with the perpetual system as final objective.

Bibliography

[1] W. Wong, "Low Power Microcontroller-based Design Techniques," 2009. [Online]. Available: http://electronicdesign.com/embedded/low-power-microcontroller-based-design-techniques

[2] A. Sedra and K. C. Smith, *Microelectronic Circuits*, 6th ed. Oxford University Press Inc, 2010.

[3] B. Ivey and Microchip Technology Inc, "AN1416, Low-Power Design Guide," 2011.

[4] Atmel Corporation, "8/16-bit Atmel XMEGA D4 Microcontroller," 2014.

[5] Microchip Technology Inc, "PIC24FJ128GA204 FAMILY Datasheet," 2014.

[6] ——, "Section 39. Power-Saving Features with Deep Sleep," 2009.

[7] Silicon Labs, "C8051F93x-C8051F92x," 2013.

[8] A. M. Holberg and A. Saetre, "Innovative Techniques for Extremely Low Power Consumption with 8-bit Microcontrollers," 2006.

[9] Texas Instruments Inc, "MSP430FR698x, MSP430FR598x Mixed-Signal Microcontrollers," 2014.

[10] ——, "MSP430x09x Family User ' s Guide," 2010.

[11] P. Thanigai and Texas Instruments Inc, "FRAMs as alternatives to flash memory in embedded designs." [Online]. Available: http://www.embedded.com/design/mcus-processors-and-socs/4390688/2/FRAMs-as-alternatives-to-flash-memory-in-embedded-designs

[12] FUJITSU SEMICONDUCTOR LIMITED, "FUJITSU Semiconductor FRAM," 2014.

[13] G. Allen, C. Stanfield, and G. Zheng, "Ferroelectric Random Access Memory," University of Michigan, Tech. Rep., 2012.

[14] K. Mikhaylov and J. Tervonen, "Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers," *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, May 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6208716

[15] J. Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, 2012.

[16] Texas Instruments Inc, "ULP Advisor - Texas Instruments," 2013. [Online]. Available: http://processors.wiki.ti.com/index.php/ULP_Advisor

[17] B. Finch, "SLAA603 Case Study," 2013. [Online]. Available: http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=slaa603&fileType=zip

[18] M. Amin Karami and D. J. Inman, "Powering pacemakers from heartbeat vibrations using linear and nonlinear energy harvesters," *Applied Physics Letters*, vol. 100, no. 4, p. 042901, Jan. 2012. [Online]. Available: http://scitation.aip.org/content/aip/journal/apl/100/4/10.1063/1.3679102

[19] B. Coxworth, "Battery-less device powers a pacemaker using heartbeats," 2014. [Online]. Available: http://www.gizmag.com/wristwatch-pacemaker/33624/

[20] A. Harb, "Energy harvesting: State-of-the-art," *Renewable Energy*, vol. 36, no. 10, pp. 2641–2654, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.renene.2010.06.014

[21] T. J. Kazmierski and S. Beeby, *Energy Harvesting Systems - Principles, Modeling and Applications*, T. J. Kazmierski and S. Beeby, Eds. Springer, 2011.

[22] R. Vullers, R. van Schaijk, I. Doms, C. Van Hoof, and R. Mertens, "Micropower energy harvesting," *Solid-State Electronics*, vol. 53, no. 7, pp. 684–693, Jul. 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0038110109000720

[23] S. Beeby and N. White, *Energy Harvesting for Autonomous Systems.* Artech House, 2010.

[24] A. Nimo, D. Grgić, and L. M. Reindl, "Ambient electromagnetic wireless energy harvesting using multiband planar antenna," in *9th International Multi-Conference on Systems, Signals and Devices, SSD 2012 - Summary Proceedings*, vol. 07, 2012. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6198036

[25] D. Hoffmann, A. Willmann, R. Göpfert, P. Becker, B. Folkmer, and Y. Manoli, "Energy Harvesting from Fluid Flow in Water Pipelines for Smart Metering Applications," *Journal of Physics: Conference Series*, vol. 476, p. 012104, 2013. [Online]. Available: http://stacks.iop.org/1742-6596/476/i=1/a=012104?key=crossref.fcc36bf8d6573535f11242b5bbd0df01

[26] C. Adkins, *Equilibrium Thermodynamics*, 3rd ed. Cambridge University Press, 1983.

[27] J. Caniou, *Passive Infrared Detection: Theory and Applications.* Springer Science + Business Media, LLC, 1999.

[28] A. Rogalski, *Infrared Detectors*, 2nd ed. CRC Press, 2011.

[29] A. Daniels, *Field Guide to Infrared Systems, Detectors, and FPAs*, 2nd ed., J. E. Greivenkamp, Ed. SPIE Press, 2010.

[30] S. Engelberg, *Random Signals and Noise: A Mathematical Introduction.* CRC Press, 2007.

[31] A. Moreira, R. Valadas, and A. de Oliveira Duarte, "Performance of infrared transmission systems under ambient light interference," *IEE Proc.-Optoelectron*, vol. 143, no. 6, 1996. [Online]. Available: http://www3.dsi.uminho.pt/adriano/publications/IEE-ProcJ96.pdf

[32] T. Agarwal, "What is a PIR Motion Sensor: PIR IC Working, Features and Applications." [Online]. Available: http://www.elprocus.com/pir-sensor-basics-applications/

[33] I. Kamal, "Infra-Red Proximity Sensor Part 2 — IKALOGIC," 2008. [Online]. Available: http://www.ikalogic.com/infra-red-proximity-sensor-part-2/

[34] Vishay Semiconductors, "TSAL6100," 2014. [Online]. Available: http://www.vishay.com/docs/81009/tsal6100.pdf

[35] ——, "TSSP58P38," 2014. [Online]. Available: http://www.vishay.com/docs/82476/tssp58p38.pdf

[36] SHARP Corporation, "SHARP GP2Y0A21YK0F," 2006. [Online]. Available: http://www.sharpsma.com/webfm_send/1489

[37] Texas Instruments Inc, "TMP006," 2011. [Online]. Available: http://www.ti.com/lit/ds/sbos518c/sbos518c.pdf

[38] Hamamatsu, "T11264 08 evaluation module," 2011.

Appendix A

Inefficient Code Version

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2014, Texas Instruments Incorporated
 * All rights reserved.
 */
//******************************************************************************
//  MSP430FR59xx EnergyTrace Demo- High Energy Consumption Code
//
//   Description: This code is intentionally written inefficiently in order
//   to use an unnecessary amount of energy. The ULP Advisor and EnergyTrace
//   are used to help identify the problem areas in the code to point out
//   where changes can be made to increase efficiency.
//
//   About every second, an ADC temperature sample is taken and the degrees
//   Celsius and Fahrenheit are found using floating point calculations.
//   The results are printed and transmitted through the UART.
//
//   B. Finch
//   Texas Instruments Inc.
//   June 2013
//   Built with Code Composer Studio V5.5.0.00039
//******************************************************************************

#include <stdio.h>
#include <msp430.h>
#include <stdint.h>

#define CAL_ADC_12T30  (*((uint16_t *)0x1A1A)) // Temperature Sensor
    Calibration-30 C 1.2V ref
#define CAL_ADC_12T85  (*((uint16_t *)0x1A1C)) // Temperature Sensor
    Calibration-85 C 1.2V ref

void UART_print(char *string);

int main(void) {
    char resultC[50];
    char resultF[50];
    float temp;
    float IntDegF;
    float IntDegC;

    WDTCTL = WDTPW | WDTHOLD;                 // Stop watchdog timer
```

```
// Configure used port pins
P2SEL1 |= BIT1 | BIT0;                          // Configure UART pins
P2SEL0 &= ~(BIT1 | BIT0);                       // Configure UART pins
PM5CTL0 &= ~LOCKLPM5;                           // Disable GPIO power-on default
    high-impedance mode
                                                // to activate previously
                                                    configured port settings
// Configure ADC12
ADC12CTL0 = ADC12SHT0_2 | ADC12ON;              // 16 ADC12CLKs, ADC ON
ADC12CTL1 = ADC12SHP | ADC12SSEL_2 | ADC12CONSEQ_0; // s/w trigger, MCLK,
    single ch/conv
ADC12CTL2 = ADC12RES__12BIT;                    // 12-bit conversion results
ADC12CTL3 = ADC12TCMAP;                         // temp sensor selected for ADC
    input channel A30
ADC12MCTL0 = ADC12VRSEL_1 | ADC12INCH_30; // ADC input ch A30 => temp sense

// Configure internal reference
while(REFCTL0 & REFGENBUSY);                    // If ref generator busy, WAIT
REFCTL0 |= REFVSEL_0 | REFGENOT | REFON;  // Select internal ref = 1.2V,
    reference on
__delay_cycles(400);                            // Delay for Ref to settle

// Configure 1sec Timer
TA0CTL = TASSEL__SMCLK | ID__8 | MC__UP | TACLR | TAIE; // SMCLK / 8, up
    mode, clear timer
TA0EX0 = TAIDEX_7;                              // (SMCLK / 8) / 8 ~ 15.625 kHz.
    Default SMCLK: 1MHz
TA0CCR0 = 0x3D09;                               // ~1 sec

// Configure UART
UCA0CTLW0 |= UCSSEL__SMCLK | UCSWRST;           // No parity, LSB first, 8-bit
    data, 1 stop
UCA0BRW = 6;                                    // Baud rate register prescale.
    Configure 9600 baud
UCA0MCTLW |= 0x2081;                            // UCBRS = 0x20, UCBRF = 8;
    UCOS16 = 1
UCA0CTLW0 &= ~UCSWRST;                          // Enable eUSCI_A

__no_operation();                               // SET A BREAKPOINT HERE

while(1)
{
    if(TA0IV == TA0IV_TAIFG)                    // Poll the timer overflow
        interrupt status
    {
        ADC12CTL0 |= ADC12ENC | ADC12SC;        // Sampling and conversion start
        while(ADC12IFGR1 & ~ADC12IFG30);        // Wait for the conversion to
            complete
        while(ADC12CTL1 & ADC12BUSY);
        temp = ADC12MEM0;

        // Temperature in Celsius:
        IntDegC = (temp - CAL_ADC_12T30) * (85.0-30.0)/(CAL_ADC_12T85-
            CAL_ADC_12T30) + 30.0;

        // Temperature in Fahrenheit:
```

```
            IntDegF = 9.0*IntDegC/5.0+32.0;

            sprintf(resultC, "%.1f Degrees Celsius\r\n", IntDegC);
            sprintf(resultF, "%.1f Degrees Fahrenheit\r\n", IntDegF);

            UART_print(resultC);                // Send the temperature
                information through the
            UART_print(resultF);                // backchannel UART

            while(UCA0STATW & UCBUSY);
            __no_operation();                   // For debugger
        }
    }
}

void UART_print(char *string)                   // Send a zero-terminated string
    through the UART
{
    char byte = *string++;
    while(byte != 0)
    {
        while(!(UCA0IFG & UCTXIFG));            // Wait until TX buffer ready
        UCA0TXBUF = byte;                       // Send the next byte of info
        byte = *string++;                       // Get the next character to send
    }
}
```

Appendix B

Most Efficient Code Version

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2014, Texas Instruments Incorporated
 * All rights reserved.
 */
//******************************************************************************
//  MSP430FR59xx EnergyTrace Demo- Low Energy Consumption Code
//
//   Description: This code has been optimized with the help of EnergyTrace
//   for minimal energy consumption.
//
//   About every second, an ADC temperature sample is taken and the degrees
//   Celsius and Fahrenheit are calculated. The results are transmitted
//   through the UART.
//
//   B. Finch
//   Texas Instruments Inc.
//   June 2013
//   Built with Code Composer Studio V5.5.0.00039
//******************************************************************************

#include <msp430.h>
#include <stdio.h>
#include <stdint.h>

#define CAL_ADC_12T30  (*((uint16_t *)0x1A1A)) // Temperature Sensor
    Calibration-30 C 1.2V ref
#define CAL_ADC_12T85  (*((uint16_t *)0x1A1C)) // Temperature Sensor
    Calibration-85 C 1.2V ref
const char cel[] = " Degrees Celsius\r\n";
const char fah[] = " Degrees Fahrenheit\r\n";


void UART_print(char *string, char type);

/* The function below assumes that the value to be converted is ten times the
    desired value. As a
 * result, the accuracy can be to the tenths place (despite the absence of
    floating point
 * variables). This implementation uses built-in functions, rather than "divide
    " and "modulo".
 */
```

```
char* rawToAsciiString(int16_t input);

char byte = 0;
int main(void) {

  int16_t temp;
  int32_t IntDegF;
  int32_t IntDegC;

    WDTCTL = WDTPW | WDTHOLD;                   // Stop watchdog timer

    // Configure Port Pins as Output Low. Clear all port interrupt flags.
    PAOUT = 0; PBOUT = 0; PJOUT = 0;
    PADIR = 0xFFFF; PBDIR = 0xFFFF; PJDIR = 0xFF;
    PAIFG = 0; PBIFG = 0;

    // Configure used port pins
    P2SEL1 |= BIT1 | BIT0;                      // Configure UART pins
    P2SEL0 &= ~(BIT1 | BIT0);                   // Configure UART pins
    PJSEL0 |= BIT4 | BIT5;                      // XT1 Setup
    PM5CTL0 &= ~LOCKLPM5;                       // Disable GPIO power-on default
        high-impedance mode
                                                // to activate previously
                                                    configured port settings
    // Configure Clock System
    CSCTL0_H = 0xA5;                            // CS password
    CSCTL2 = SELA__LFXTCLK;                     // ACLK sourced from LFXT
    CSCTL3 = DIVA__1;                           // No division
    CSCTL4 |= LFXTDRIVE_3 | SMCLKOFF | VLOOFF; // Highest crystal drive setting
        . MAY CHANGEEEEE SMCLK turned off.
    CSCTL4 &= ~LFXTOFF;                         // Turn on LFXT

    do
    {
      CSCTL5 &= ~(LFXTOFFG | HFXTOFFG);         // Clear XT1 fault flag
      SFRIFG1 &= ~OFIFG;
    }while (SFRIFG1 & OFIFG);                   // Test oscillator fault flag

    // Configure ADC12
    ADC12CTL0 = ADC12SHT0_2 | ADC12ON;          // 16 ADC12CLKs; ADC ON
    ADC12CTL1 = ADC12SHP | ADC12SSEL_1 | ADC12CONSEQ_0; // s/w trigger, ACLK,
        single ch/conv
    ADC12CTL2 = ADC12RES__12BIT;                // 12-bit conversion results.
    ADC12CTL3 = ADC12TCMAP;                     // temp sensor selected for ADC
        input channel A30
    ADC12MCTL0 = ADC12VRSEL_1 | ADC12INCH_30; // ADC input ch A30 => temp sense
    ADC12IER0 |= ADC12IE0;                      // Enable A30 interrupt

    // Configure internal reference
    while(REFCTL0 & REFGENBUSY);                // If ref generator busy, WAIT
    REFCTL0 |= REFVSEL_0 | REFGENOT | REFON;   // Select internal ref = 1.2V,
        reference on

    // Configure Timer
    TA0CTL = TASSEL__ACLK | MC__UP | TACLR;     // ACLK, up mode, clear timer.
    TA0CCR0 = 131;                              // ~0.4ms
```

```
    TA0CCTL0 |= CCIE;                         // Capture/compare interrupt
        enable.
    __bis_SR_register(LPM3_bits | GIE);       // Enter LPM3. Delay for Ref to
        settle.
    TA0CCR0 = 0x8000;                         // Change timer delay to ~1 sec.

    // Configure UART
    UCA0CTLW0 |= UCSSEL__ACLK | UCSWRST;      // No parity, LSB first, 8-bit
        data, 1 stop
    UCA0BRW = 3;                              // Baud rate register prescale.
        Configure 9600 baud
    UCA0MCTLW = 0x9200;                       // UCBRS = 0x92, UCBRF = -- (don'
        t care); UCOS16 = 0
    UCA0CTLW0 &= ~UCSWRST;                    // Enable eUSCI_A

    __no_operation();                         // SET BREAKPOINT HERE

    while(1)
    {
        __bis_SR_register(LPM3_bits | GIE);   // Enter LPM3, wait for ~1sec
            timer

        ADC12CTL0 |= ADC12ENC | ADC12SC;      // Sampling and conversion start
        __bis_SR_register(LPM3_bits | GIE);   // Wait for conversion to
            complete
        temp = ADC12MEM0;                     // 'temp' = the raw ADC
            temperature conversion result
        //__bic_SR_register(GIE);

        // Temperature in Celsius, multiplied by 10:
        IntDegC = ((temp - CAL_ADC_12T30)*10*(85-30)*10/((CAL_ADC_12T85-
            CAL_ADC_12T30)*10) + 30*10);

        // Temperature in Fahrenheit, multiplied by 10:
        IntDegF = 9*IntDegC/5+320;

        UART_print(rawToAsciiString(IntDegC), 'C'); // Send temperature
            information through UART
        UART_print(rawToAsciiString(IntDegF), 'F');
    }
}

char* rawToAsciiString(int16_t input)         //conversion algorithm which used
     built-in functions
{                                             // (rather than "divide" and "
    modulo")
    uint16_t i;
    uint16_t bcd;
    static char result[6];

    if (input < 0) input = -input;

    for (i = 16, bcd = 0; i; i--)
    {
        bcd = __bcd_add_short(bcd, bcd);
        if (input & 0x8000)
            bcd = __bcd_add_short(bcd,1);
```

```
        input <<= 1;
    }

    for (i = 4; i > 0; i--)
    {
        result[i-1] = 0x30 | ((bcd>>((4-i)*4))&0xF);
    }
    result[4] = result[3];                      // Move 10ths place
    result[3] = '.';                            // Insert decimal point

    return &result[0];
}

void UART_print(char *string, char type)        // Send a zero-terminated string
    through the UART
{
    byte = *string++;
    char count = 0;


    while (byte == 0x30)                        // Don't print leading zeros.
    {
        byte = *string++;
        count++;
    }

    while(count != 5)
    {
        UCA0IE |= UCTXIE;                       // Enable UART TX interrupt
        __bis_SR_register(LPM3_bits | GIE);     // Wait until TX buffer ready
        byte = *string++;
        count++;
    }
    count = 0;
    if (type == 'C')
    {
        while(cel[count] != 0)
        {
            byte = cel[count++];                // Send the next byte of info
            UCA0IE |= UCTXIE;                   // Enable UART TX interrupt
            __bis_SR_register(LPM3_bits | GIE); // Wait until TX buffer ready
        }
    }
    else if (type == 'F')
    {
        while(fah[count] != 0)
        {
            byte = fah[count++];                // Send the next byte of info
            UCA0IE |= UCTXIE;                   // Enable UART TX interrupt
            __bis_SR_register(LPM3_bits | GIE); // Wait until TX buffer ready
        }
    }
    UCA0IE &= ~UCTXIE;
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
```

```
{
  switch(__even_in_range(ADC12IV,76))
  {
    case 12:                                  // Vector 12:  ADC12MEM0
        ADC12IFGR0 &= ~ADC12IFG0;             // Clear interrupt flag
        __bic_SR_register_on_exit(LPM3_bits); // Exit active CPU
        break;
    default: break;
  }
}

// Timer0_A3 Interrupt Vector (TAIV) handler
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{
    __bic_SR_register_on_exit(LPM3_bits);     // Exit active CPU
}

#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
  switch(__even_in_range(UCA0IV,USCI_UART_UCTXCPTIFG))
  {
    case USCI_NONE: break;
    case USCI_UART_UCRXIFG: break;
    case USCI_UART_UCTXIFG:
        UCA0TXBUF = byte;                     // Send the next byte of info
        UCA0IE &= ~UCTXIE;                    // Disable UART TX interrupt
        __bic_SR_register_on_exit(LPM3_bits); // Exit active CPU
        break;
    case USCI_UART_UCSTTIFG: break;
    case USCI_UART_UCTXCPTIFG: break;
  }
}
```

Appendix C

Implemented Code

```
#include <asf.h>
#include <delay.h>

#define STATLED     IOPORT_CREATE_PIN(PORTD, 4)
#define IROUT       IOPORT_CREATE_PIN(PORTD, 7)

inline void SleepTimer(void);
inline void configCPUspeed(void);
inline void configPorts(void);
inline void InfraredTimer(void);
inline void DebounceTimer(void);
inline void disableDebounceTimer(void);
inline void disableSleepTimer(void);
inline void disableInfraredTimer(void);
inline void goToSleep(void);
inline void wakeUp(void);
inline void enableValve(void);
inline void disableValve(void);
inline void idleValve(void);
inline void enableLED(void);
inline void disableLED(void);

volatile unsigned char validated_high = 0;
volatile unsigned char previously_detected = 0;
volatile unsigned char previously_activated = 0;
volatile unsigned char validated_low = 0;
volatile unsigned char debounce_counter = 0;
volatile unsigned char wakeup_control = 1;
unsigned char delta = 5; //[multiply by 10 to avoid floating point calculations
    ]

int main (void)
{
  configCPUspeed();
  configPorts();
  SleepTimer();
  InfraredTimer(); // ENABLE INFRARED
  DebounceTimer(); // READ INFRARED
  sei();
  while(1)
  {
```

```
  }
}

inline void configCPUspeed(void)
{
  /*
  The CLKPCE bit must be written to logic one to enable change of the CLKPS
      bits. The CLKPCE
  bit is only updated when the other bits in CLKPR are simultaneously written
      to zero. CLKPCE is
  cleared by hardware four cycles after it is written or when CLKPS bits are
      written.
  */
  CLKPR = (1 << CLKPCE); // enable a change to CLKPR
  CLKPR = (1 << CLKPS2); // 8000000 / 16 = 500 kHz
}
inline void configPorts(void)
{
  // write one to configure as output
  DDRB = 0xC7; // no not touch the MOSI MISO and SCK pins
  PORTB = 0x00;

  DDRC = 0x3F; // no not touch the RST pin
  PORTC = 0x00;
  DDRD = 0x73; // PD3 and PD2 as inputs and PD7
  PORTD = 0x00;
}
inline void SleepTimer(void) // T2
{
  // 1 Hz minimum frequency
  // 1024 prescaling
  TCCR2A = (0 << COM2A1) | (0 << COM2A0) | (0 << COM2B1) | (0 << COM2B0) | (1
      << WGM21) | (0 << WGM20);
  TCCR2B = (0 << WGM22) | (1 << CS22) | (1 << CS21) | (1 << CS20);
  TCNT2 = 0x00;
  OCR2A = 121; // 2Hz
  TIMSK2 |= (1 << OCIE2A); // interrupt enable
}
inline void InfraredTimer(void) // T1 38 kHz timer
{
  /*
  A frequency (with 50% duty cycle) waveform output in fast PWM mode can be
      achieved by setting
  OC1A to toggle its logical level on each compare match (COM1A1:0 = 1). This
      applies only
  if OCR1A is used to define the TOP value (WGM13:0 = 15).
  */
  // IMPORTANT: THERE IS A /2 DIVIDER. BEST ACHIEVABLE FREQUENCY IS 35.7 kHz
  TCCR1A = (0 << COM1A1) | (1 << COM1A0) | (0 << COM1B1) | (0 << COM1B0) |  (1
      << WGM11) | (1 << WGM10);
  // PWM mode.. toggle OC1A on compare match
  //TCCR1B =  (0 << ICNC1) | (0 << ICES1) | (1 << WGM13) | (1 << WGM12) | (0 <<
       CS12) | (0 << CS11) | (1 << CS10);
  TCCR1B = 0x19;
  // no prescaling
  TCNT1H = 0x0;
```

```
  TCNT1L = 0x0;
  OCR1AL = 6;
  OCR1AH = 0;
  TIMSK1 = 0x00;
  DDRD|= 0x02;
}
inline void DebounceTimer(void) //T0 debouncer timer
{
  // Debouncer Timer
  // Configured with a 2550 Hz frequency
  // CTC mode with OCR0A interrupt
  // no pin change
  // no prescaler
  TCCR0A = (0 << COM0A1) | (0 << COM0A0) | (0 << COM0B1) | (0 << COM0B0) | (1
      << WGM01) | (0 << WGM00);
  TCCR0B = (0 << WGM02) | (0 << CS02) | (0 << CS01) | (1 << CS00);
  TCNT0 = 0x00;
  OCR0A = 98;
  TIMSK0 = (1 << OCIE0A); // Output compare match A interrupt enable
}
ISR(TIMER2_COMPA_vect) // Sleeper Routine
{
  if (wakeup_control == 1)
  {
    wakeUp();
    InfraredTimer();
    DebounceTimer();

        if (validated_high == 1) // ENABLE INFRARED -> DETECTION? (YES)
        {
          if ((previously_activated == 1) & (validated_high == 1)) // ENABLE
              INFRARED -> DETECTION? (YES) -> PREVIOUSLY ACTIVATED? (YES)
          {
            //disableDebounceTimer(); // ENABLE INFRARED -> DETECTION? (YES) ->
                PREVIOUSLY ACTIVATED? (YES) -> DISABLE INFRARED
            //disableInfraredTimer(); //

          } // ENABLE INFRARED -> DETECTION? (YES) -> PREVIOUSLY ACTIVATED? (
              YES)

          if ((previously_activated == 0) & (validated_high == 1)) // ENABLE
              INFRARED -> DETECTION? (YES) -> PREVIOUSLY ACTIVATED? (NO)
          {
            enableLED();
            enableValve();       // ENABLE INFRARED -> DETECTION? (YES) ->
                PREVIOUSLY ACTIVATED? (NO) -> ACTIVATE VALVE
            disableDebounceTimer(); // ENABLE INFRARED -> DETECTION? (YES) ->
                PREVIOUSLY ACTIVATED? (NO) -> ACTIVATE VALVE -> DISABLE INFRARED
            disableInfraredTimer(); //
            delay_ms(50);
            idleValve();
            previously_activated = 1;
          } // ENABLE INFRARED -> DETECTION? (YES) -> PREVIOUSLY ACTIVATED? (NO
              )

          previously_detected = 1;
        } // ENABLE INFRARED -> DETECTION? (YES)
```

```
        if (validated_low == 1)  // ENABLE INFRARED -> DETECTION? (NO)
        {
          if ((previously_detected == 1) & (previously_activated == 1) & (
              validated_low == 1)) // ENABLE INFRARED -> DETECTION? (NO) ->
              PREVIOUSLY DETECTED? (YES)
          {
            disableLED();
            disableValve(); // ENABLE INFRARED -> DETECTION? (NO) -> PREVIOUSLY
                DETECTED? (YES) -> DEACTIVATE VALVE
            //disableDebounceTimer(); // ENABLE INFRARED -> DETECTION? (NO) ->
                PREVIOUSLY DETECTED? (YES) -> DEACTIVATE VALVE -> DISABLE
                INFRARED
            //disableInfraredTimer(); //
            delay_ms(50);
            idleValve();
            previously_activated = 0;
          } // ENABLE INFRARED -> DETECTION? (NO) -> PREVIOUSLY DETECTED?

          if ((previously_detected == 0) & (previously_activated == 0) & (
              validated_low == 1))
          {
            disableLED();
            // just do nothing. if nothing is detected it keeps jumping here
                doing nothing.
          }
          previously_detected = 0;
          previously_activated = 0;

        } // ENABLE INFRARED -> DETECTION? (NO)

    wakeup_control = 0;
  }
  else
  {
    wakeup_control = 1;
    disableDebounceTimer();
    disableInfraredTimer();
    goToSleep();
  }
}
ISR(TIMER0_COMPA_vect) // Debounce routine
{
  // Read Sensor PIN: PD7
  // 7654.3210
  // 1000.0000
  if (!ioport_pin_is_low(IROUT))
  {
    // decrement debouncer
    if (debounce_counter > 0)
    {
      debounce_counter--;
      validated_low = 0;
      validated_high = 0; // make sure they are both zero or opposite
    }
    else
    {
```

```
      debounce_counter = 0;
      validated_low = 1; // make sure they are both zero or opposite
      validated_high = 0;
    }
  }
  else // if(aux == 0x00)
  {
    // increment debouncer
    if (debounce_counter < 255)
    {
      debounce_counter++;
      validated_low = 0;
      validated_high = 0;
    }
    else
    {
      debounce_counter = 255;
      validated_high = 1;
      validated_low = 0;
    }
  }
}
inline void disableSleepTimer(void)
{
  TCCR2B &= (0 << CS22) | (0 << CS21) | (0 << CS20);
  TCNT2 = 0x00;
}
inline void enableValve(void)
{
  // PC1: CTRLA
  // PC0: CTRLB
  PORTC &= 0xFC;
  PORTC |= 0x02;
}
inline void disableValve(void)
{
  // PC1: CTRLA
  // PC0: CTRLB
  PORTC &= 0xFC;
  PORTC |= 0x01;
}
inline void idleValve(void)
{
  // PC1: CTRLA
  // PC0: CTRLB
  PORTC &= 0xFC;
}
inline void goToSleep(void)
{
  PRR = (1 << PRTWI) | (1 << PRTIM0) | (1 << PRTIM1) | (1 << PRSPI) | (1 <<
      PRUSART0) | (1 << PRADC);
  SMCR = (0 << SM2) | (1 << SM1) | (1 << SM0); // power save
  SMCR |= (1 << SE); // sleep enable
}
inline void wakeUp(void)
{
```

```
  PRR = (1 << PRTWI) | (0 << PRTIM0) | (0 << PRTIM1) | (1 << PRSPI) | (1 <<
      PRUSART0) | (1 << PRADC);
  SMCR = 0x00; // idle
}
inline void disableDebounceTimer(void)
{
  TCCR0B &= (0 << CS02) | (0 << CS01) | (0 << CS00);
  TCNT0 = 0x00;
}
inline void disableInfraredTimer(void)
{
  TCCR1B &= (0 << CS12) | (0 << CS11) | (0 << CS10);
  TCNT1L = 0x00;
  TCNT1H = 0x00;
}
inline void enableLED(void)
{
  PORTD |= 0x10;
}
inline void disableLED(void)
{
  PORTD &= 0xEF;
}
```