Helsinki Metropolia University of Applied Sciences
Degree Programme in Information Technology

David Álvarez Sánchez

Design of a remote controlled hovercraft

Bachelor's Thesis. 17th April 2009
Supervisor: Antti Piironen, Principal Lecturer
Language advisor: Marianne Kiekara, Lecturer

PROYECTO FIN DE CARRERA

TEMA: Sistema de control de velocidad y dirección

TÍTULO: Diseño de un aerodeslizador con control remoto

AUTOR: David Álvarez Sánchez

TUTOR: Antti Piironen

DEPARTAMENTO: Sistemas embebidos

CENTRO DE LECTURA: Helsinki Metropolia University of applied sciences

FECHA DE LECTURA: 14 de Mayo de 2009

CALIFICACIÓN:

RESUMEN DEL PROYECTO:

Este informe trata el diseño, desarrollo y construcción de un aerodeslizador de pequeño tamaño, equipado con control remoto que permite al usuario actuar sobre la velocidad y dirección del mismo. Este proyecto podrá ser utilizado en un futuro como base para el desarrollo de aplicaciones más complejas.

Un aerodeslizador es un medio de transporte cuyo chasis se eleva sobre el suelo por medio de un motor impulsor que hincha una falda colocada en la parte inferior del mismo. Además, uno o más motores se colocan en la parte trasera del vehículo para propulsarlo. El hecho de que el aerodeslizador no este en contacto directo con la tierra, hace que pueda moverse tanto por tierra como sobre el agua o hielo y que sea capaz de superar pequeños obstáculos. Por otra parte, este hecho se convierte a su vez en un problema debido a que su fuerza de rozamiento al desplazarse es muy pequeña, lo que

provoca que sea muy difícil de frenar, y tienda a girar por sí mismo debido a la inercia del movimiento y a las fuerzas provocadas por las corrientes de aire debajo del chasis. Sin embargo, para este proyecto no se ha colocado una falda debajo del mismo, debido a que su diseño es bastante complicado, por lo tanto la fricción con el suelo es menor, aumentando los problemas detallados con anterioridad.

El proyecto consta de dos partes, mando a distancia y aerodeslizador, que se conectan a través de antenas de radiofrecuencia (RF). El diseño y desarrollo de cada una ha sido realizado de manera separada exceptuando la parte de las comunicaciones entre ambas.

El mando a distancia se divide en tres partes. La primera está compuesta por la interfaz de usuario y el circuito que genera las señales analógicas correspondientes a sus indicaciones. La interfaz de usuario la conforman tres potenciómetros: uno rotatorio y dos deslizantes. El rotatorio se utiliza para controlar la dirección de giro del aerodeslizador, mientras que cada uno de los deslizantes se emplea para controlar la fuerza del motor impulsor y del propulsor respectivamente. En los tres casos los potenciómetros se colocan en el circuito de manera que actúan como divisores de tensión controlables. La segunda parte se compone de un microcontrolador de la familia PSoC. Esta familia de microcontroladores se caracteriza por tener una gran adaptabilidad a la aplicación en la que se quieran utilizar debido a la posibilidad de elección de los periféricos, tanto analógicos como digitales, que forman parte del microcontrolador. Para el mando a distancia se configura con tres conversores A/D que se encargan de transformar las señales procedentes de los potenciómetros, tres amplificadores programables para trabajar con toda la escala de los conversores, un LCD que se utiliza para depurar el código en C con el que se programa y un módulo SPI que es la interfaz que conecta el microcontrolador con la antena. Además, se utilizan cuatro pines externos para elegir el canal de transmisión de la antena. La tercera parte es el módulo transceptor de radio frecuencia (RF) QFM-TRX1-24G, que en el mando a distancia funciona como transmisor. Éste utiliza codificación Manchester para asegurar bajas tasas de error. Como alimentación para los circuitos del mando a distancia se utilizan cuatro pilas AA de 1,5 voltios en serie.

En el aerodeslizador se pueden distinguir cinco partes. La primera es el módulo de comunicaciones, que utiliza el mismo transceptor que en el mando a distancia, pero esta vez funciona como receptor y por lo tanto servirá como entrada de datos al sistema haciendo llegar las instrucciones del usuario. Este módulo se comunica con el siguiente, un microcontrolador de la familia PSoC, a través de una interfaz SPI. En este caso el microcontrolador se configura con: un modulo SPI, un LCD utilizado para depurar el código y tres módulos PWM (2 de 8 bits y uno de 16 bits) para controlar los motores y el servo del aerodeslizador. Además, se utilizan cuatro pines externos para seleccionar el canal de recepción de datos. La tercera y cuarta parte se pueden considerar conjuntamente. Ambas están compuestas por el mismo circuito electrónico basado en transistores MOSFET. A la puerta de cada uno de los transistores llega una señal PWM de 100 kilohercios que proviene del microcontrolador, que se encarga de controlar el modo de funcionamiento de los transistores, que llevan acoplado un disipador de calor para evitar que se quemen. A su vez, los transistores hacen funcionar al dos ventiladores, que actúan como motores, el impulsor y el propulsor del aerodeslizador. La quinta y última parte es un servo estándar para modelismo. El servo está controlado por una señal PWM, en la que la longitud del pulso positivo establece la posición de la cabeza del servo, girando en uno u otra dirección según las instrucciones enviadas desde el mando a distancia por el usuario. Para el aerodeslizador se han utilizado dos fuentes de alimentación diferentes: una compuesta por 4 pilas AA de 1,5 voltios en serie que alimentarán al microcontrolador y al servo, y 4 baterías de litio recargables de 3,2 voltios en serie que alimentan el circuito de los motores.

La última parte del proyecto es el montaje y ensamblaje final de los dispositivos. Para el chasis del aerodeslizador se ha utilizado una cubierta rectangular de poli-estireno expandido, habitualmente encontrado en el embalaje de productos frágiles. Este material es bastante ligero y con una alta resistencia a los golpes, por lo que es ideal para el propósito del proyecto. En el chasis se han realizado dos agujeros: uno circular situado en el centro del mismo en el se introduce y se ajusta con pegamento el motor impulsor, y un agujero con la forma del servo, situado en uno del los laterales estrechos del rectángulo, en el que se acopla el mismo. El motor propulsor está adherido al cabezal giratorio del servo de manera que rota a la vez que él, haciendo girar al aerodeslizador. El resto de circuitos electrónicos y las baterías se fijan al chasis mediante cinta adhesiva y pegamento procurando en todo momento repartir el peso de manera homogénea por todo el chasis para aumentar la estabilidad del aerodeslizador.

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	David Álvarez Sánchez
Title	Design of a remote controlled hovercraft
Number of Pages	55
Date	17 April 2009
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Supervisor	Antti Piironen, Principal Lecturer

In this final year project a remote controlled hovercraft was designed using mainly technology that is well known by students in the embedded systems programme. This platform could be used to develop further and more complex projects.

The system was developed dividing the work into two parts: remote control and hovercraft. The hardware was of the hovercraft and the remote control was designed separately; however, the software was designed at the same time since it was needed to develop the communication system.

The result of the project was a remote control hovercraft which has a user friendly interface. The system was designed based on microprocessor technologies and uses common remote control technologies.

The system has been designed with technology commonly used by the students in Metropolia University so that it can be readily understood in order to develop other projects based on this platform.

Keywords	embedded systems, microprocessor, printed circuit board, hovercraft, antenna

Contents

Abbreviations and terms

ACV:	Air Cushion Vehicle
ADC:	Analog to Digital Converter
API:	Application Programming Interface
ASCII:	American Standard Code for Information Interchange
ASIC:	Application-Specific Integrated Circuits
BPSK:	Binary Phase-Shift Keying
CAD:	Computer-Aided Design
CPU:	Central Processing Unit
DSP:	Digital Signal Processor
GPIO:	General Purpose Input Output
IC:	Integrated Circuit
ILO:	Internal Low Speed Oscillator
IMO:	Internal Main Oscillator
I/O:	Input/Output
ISSP:	In-System Serial Programming
KBPS:	Kilobits Per Second
LCD:	Liquid Crystal Display
LED:	Light-Emitting Diode
MIPS:	Million Instructions Per Second
NRZ:	Non Return to Zero
PC:	Personal Computer
PCB:	Printed Circuit Board
PCM:	Pulse Code Modulation
PSoC:	Programmable System-On-Chip
PWM:	Pulse-Width Modulator
RF:	Radio Frequency
SPI:	Serial Peripheral Interface
SRAM:	Static Random Access Memory
UK:	United Kingdom
USA:	United States of America

USB:	Universal Serial Bus
V:	Volts
WDT:	Watch Dog Timers

1 Introduction

This report discusses the design, development and construction of a model-size remote controlled hovercraft, which could be used in the future as a platform to develop more advanced tasks by students of Helsinki Metropolia University of Applied Sciences.

The main goal of this design was to provide the hovercraft (initially, the word hovercraft will refer to the moving part of the device) with a system to control its steering following the instructions the user provides by the remote control. This is not a simple issue due to the fact that the hovercraft 'flies' over the ground taking away almost all the friction force and therefore it is very difficult to stabilize, which increases the handicap to reach an accurate performance.

The following paper aims to explain the hardware and software design flow and of course the final construction of the hovercraft itself. Hence a student interested in this paper would normally be studying the embedded engineering major although almost every student in the information technology programme degree would be able to understand most of the topics discussed.

2 Background theory

2.1 Embedded systems

This project can be classified as an embedded systems design. An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, so they usually take a different form from general purpose computers [1]. Embedded computers consist of a programmable part together with ASICs (application-specific integrated circuits) and other standard components that interact with the environment. The programmable part is usually a microprocessor or a DSP (digital signal processor) with memory, either internal or external, where the program and data needed by the processor are stored.

The fact that it is designed for a particular purpose permits the engineers to design more specific hardware and software, which improves the final performance usually reducing both the cost and the size of the product. Furthermore some embedded systems can be considered as real-time systems, for safety or usability reasons [2].

Embedded systems are all continuously around us in every-day life forming part of consumer electronics like watches, credit cards, mobile phones, etc, and are also found in many different applications such as aerospace or automobile industry as shown in figure 1.

Figure 1. Examples of embedded systems applications [3].

2.2 Printed circuit board

When designing embedded systems it is usual to use PCB (printed circuit board) technology so that the components of the electronic design are placed on the same board and connected using copper paths. PCBs are robust and inexpensive. Although the design flow and the placement of the components is usually slower than in either wire-wrapped or point-to-point constructed circuits the final result is always much more reliable and the area used is smaller. Nowadays PCBs are designed using CAD (Computer-Aided Design) software which helps the designer in the different tasks of the design flow.

PCB design flow [4]

The first step is to do the schematic design. A schematic is a diagram in which the parts of the circuit and their interconnections are defined. The designer must be careful in selecting the parts so that they are available at a reasonable cost and in due time because it is almost impossible to use a part with a different package.

The second step is to set the design rules. Depending on the software the designer is using, this step is done in either the schematic designer or in the PCB designer. Typical design constraints are:

- Trace width
- Spacing
- Maximum and minimum net length
- Allowed delay
- Capacitance
- Terminations
- Layer restrictions
- etc

When the designer has finished working with the schematic designer, all the information is transferred to the PCB designer using a net list. This minimizes the possibility of human error and reduces documentation.

The next and last step is the PCB design. This last part can be divided into different tasks:

- *PCB board outline*: It consists in defining the edges of the board. Usually it has a simple shape and it is done with the PCB designer but for complex ones it can be imported from a mechanical cad package such as AutoCAD.
- *Component placement*: This is the most important part of the design flow and it is usually the longest one. The different components should be placed wisely so that later on the designer is able to mount and solder them. The designer also has to take care of where he puts the input and output connectors and also the test points so that they are easily reachable.
- *Routing*: At this point the designer adds the paths among the components to make the electrical connections. The difficulty of this task depends to a great extent on the circuit being designed. Digital designs tend to have fewer components than analog ones but greater pin count. A good placement always makes it easier.
- *Document text*: There are often some text entries in the design to identify the finished PCB.
- *Post processing*: It means producing the necessary documentation for the layer and surface treating when manufacturing the PCB. The CAD holds all this information in a single structure.

2.3 Hovercraft

A hovercraft or ACV (air cushion vehicle) is a means of transport whose chassis is lifted from the ground by an impeller that inflates a cushion underneath it with high pressure air. Besides, one or more engines are placed in the back part of the craft in order to propel it. The fact that the part of the vehicle which is in contact with the ground is the cushion enables the hovercraft to move not only over land but also over water or ice and even to overcome small obstacles. However, this fact is also a challenge because due to the low friction resistance they have, hovercrafts are difficult to stop and tend to drift themselves. This self-drifting is mainly caused by the inertia of the movement and the air dynamics in the cushion. The next graphic shows the main parts of a hovercraft.

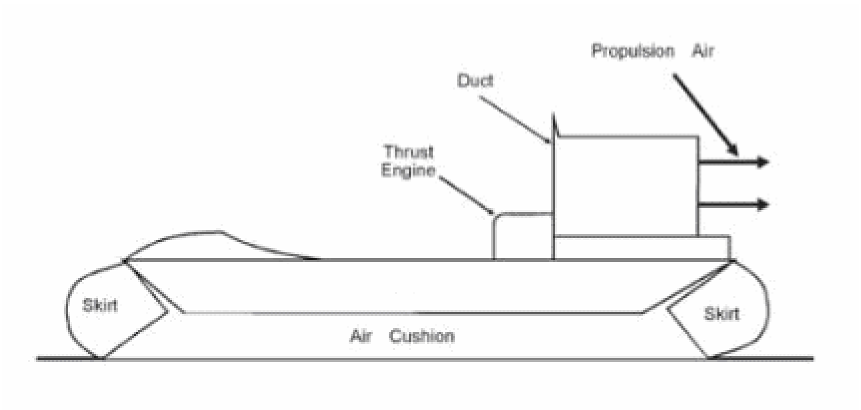

Figure 2. Schematic draw of a hovercraft.

However, the aim for the development of this project was not to mount a real hovercraft but a model-size one, anyway; the difficulties encountered to get a good performance were very similar to the ones explained right before. Besides, the model-size hovercraft was not to use a cushion in its lower part because the design of the cushion itself would have been a very difficult task, which was not a goal of this project. This introduces two main differences to the explanation given before: the hovercraft would not be able to overcome any obstacle and therefore, the friction resistance would be lower and would cause the hovercraft to be more difficult to control.

It can be said that the idea of using an ACV as a means of decreasing the friction while moving was first explored by Sir John Thornycroft, a British engineer, who in the 1870's built some experimental models on the basis of an air cushion system that would reduce the drag of water on boats and ships. Since then many different patents have been developed around this idea approaching towards what we nowadays call a hovercraft; the most important one is known as a flying boat which was widely used in the first part of 20th century.

Nevertheless it was not until 1952 that the British inventor Sir Christopher Cockerell proved a workable principle of a vehicle suspended on a cushion of air blown out under pressure making the vehicle easily movable over any surface by using simple vacuum cleaner engines. A few years later Saunders Roe, a British aircraft manufacturer, developed the first practical hovercraft, the SR-N1, presented in the first public demonstration in 1959, as can be seen in the next figure [5].

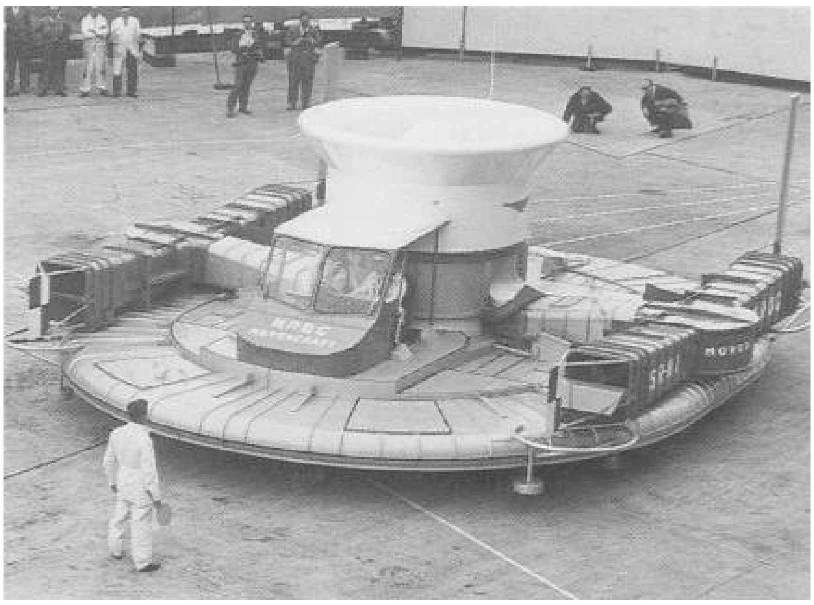

Figure 3. SR-N1 trials in June 1959 [6].

Since then, hovercrafts have had a lot of different application fields. Nowadays there are large hovercrafts used for goods and passenger transport usually in areas surrounded by lakes or with rough terrain that would make the use of trucks a waste of time. There is also an increasing number of private users who use smaller ones; some of them are homebuilt to move around large terrains or marshy areas or simply for leisure issues like racing purposes. Also, there are hovercrafts designed for military purposes. The navies of different countries have developed their own hovercrafts throughout history: SR.N5 used by the USA (United States of America) army in the Vietnam War, the LCAC-class air-cushioned landing craft actually deployed by the Japanese and USA navies, the Zubr class LCAC, which is the largest hovercraft and was developed by the Soviet Union, the Griffon 2000 TDX class ACV recently used by UK (United Kingdom) Royal Marine in Iraq or the Jingsah II class LCAC used by the People's Army Navy of China [7].

2.4 Microprocessor

As I already mentioned, in every embedded system there is usually a microprocessor. This project is going to be developed using PSoC (Programmable System-on-chip) technology by *Cypress Semiconductor*, commonly used by the students majoring in embedded systems at Metropolia University.

PSoC includes a CPU (Central Processing Unit), memory, clocks, and configurable GPIO (General Purpose Input Output) and analog and digital peripherals. The M8C CPU core is a powerful processor which speeds up to 24 MHz, providing a four MIPS (Million Instructions per Second) 8-bit Harvard architecture microprocessor. The CPU uses an interrupt controller with up to 20 vectors, to simplify programming of real time embedded events. Program execution is timed and protected using the included Sleep and WDT (Watch Dog Timers). The PSoC device incorporates flexible internal clock generators, including a 24 MHz IMO (internal main oscillator) which can also be doubled to 48 MHz for use by the digital system. The clocks, together with programmable clock dividers (as a system resource), provide the flexibility to integrate almost any timing requirement into the PSoC device. PSoC has three separate memory spaces: 1K of paged SRAM (static random access memory) for data, 16K of flash memory for program storage data and up to 2K of EEPROM (Electrically Erasable Programmable Read Only Memory) emulated using the flash memory. PSoC GPIO pins provide connection to the CPU, digital and analog resources of the device. Each pin’s drive mode may be selected from eight options, allowing great flexibility in external interfacing. Every pin is also capable of generating a system interrupt on high level, low level, and change from last read [8].

PSoC can be easily understand if we think about it as a microcontroller in which the peripherals, both analog and digital, are chosen by the programmer observing restrictions like the quantity of peripherals and their type. Also internal communication among different peripherals and the I/O pins of the chip can be configured using a graphical interface. After choosing the peripherals the programmer can interact with them using functions provided by automatically generated APIs (Application

Programming Interface) which make them easy to program. Besides, the PSoC Designer (software provided by Cypress Semiconductor) generates the start-up configuration based on the selections done. PSoC Designer supports high-level C language, due to the fact that it includes a specific compiler that supports PSoC family of devices. The figure below shows the development process using PSoC Designer.

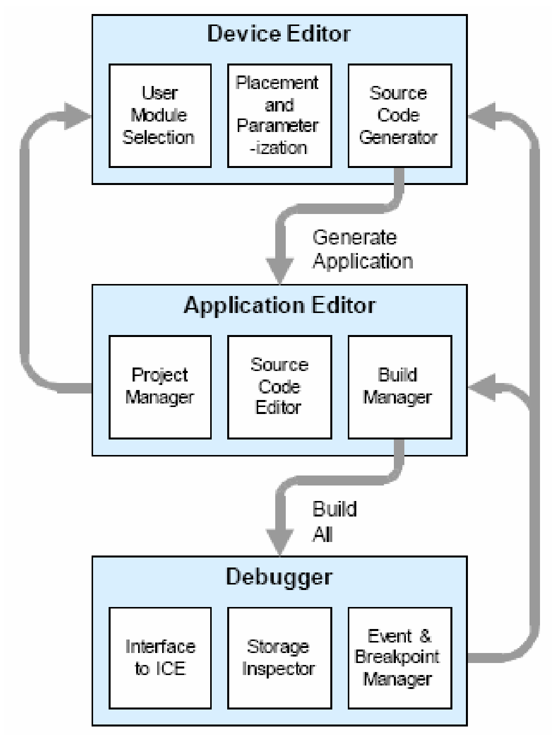

Figure 4. User module and source code development code [8].

The PSoC Designer also includes a debugger subsystem that uses high functionality ICE (In-Circuit Emulator) connected to the PC (Personal Computer) by way of an USB (universal serial bus) port, allowing the designer to test the program in a physical system while providing an internal view of the PSoC device. Debugger commands allow the designer to read and write data memory, read and write IO registers, read and write CPU registers, set and clear breakpoints, and provide program run, halt, and step control. The debugger also allows the designer to create a trace buffer of registers and memory locations of interest.

2.5 Communications

Since the hovercraft was to be remote controlled, I needed a way to communicate the remote control with the hovercraft itself. The technology used was RF (Radio Frequency) communication because the university already had some RF transceivers. These transceivers were already part of a chip developed by *Quasar UK* so no design of the antenna was needed. Besides, this module provides a *SPI* (Serial Peripheral Interface) bus to communicate with the microprocessor and *Manchester encoding* and decoding to ensure low error rates.

A Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a "four wire" serial bus, contrasting with three, two, and one wire serial buses. [9]

Manchester code (also known as Phase Encoding) is a line code in which the encoding of each data bit has at least one transition and occupies the same time. It is, therefore, self-clocking, which means that a clock signal can be recovered from the encoded data. It is a special case of BPSK (Binary Phase-Shift Keying), where the data controls the phase of a square wave carrier whose frequency is the data rate. It always has a transition in the middle of each bit period and may, depending on the information to be transmitted, have a transition at the start of the period also. The direction of the mid-bit transition indicates the type of data (1 or 0). Transitions at the period boundaries do not carry information. They exist only to place the signal in the correct state to allow the mid-bit transition. Although this allows the signal to be self-clocking, it doubles the bandwidth requirement compared to NRZ (non-return-to-zero) coding schemes [10].

3 Methods and materials

3.1 Hardware

This project had two different parts, connected by the RF antennas, the remote control and the hovercraft. The development of the two is discussed in different sections though when developing and testing the communications both parts were done at the same time.

3.1.1 Remote Control

The block diagram of the remote control, shown in figure 5, can be divided into three main parts: the first one is controlled by the user to indicate the position of the servo and the speed of both fans, the second one includes the microprocessor and the modules needed to interface with the first and the third parts, and the last one is the part responsible for the communications.

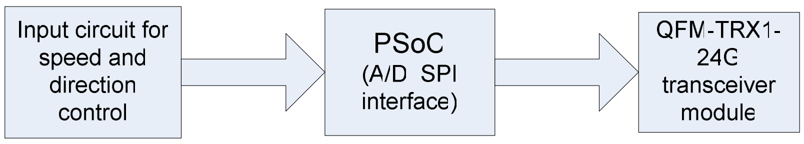
Input circuit for speed and direction control
PSoC (A/D SPI interface)
QFM-TRX1-24G transceiver module

Figure 5. Basic block diagram of the remote control.

Since I was working with PSoC technology the first decision I had to take was to determine which PSoC part among the ones used by the university had enough analog and digital blocks to meet the specifications of my design. The remote control needed a big amount of analog blocks since several ADCs (Analog to Digital Converter) were necessary in this module. I took the decision of working with the developing board *29466* that is usually used by the students majoring embedded engineering and therefore it will be familiar in case of any further modification by students of Metropolia University.

This board included the PSoC part *CY8C29466–24PX,* as well as a circuit for a reset button, a circuit for controlling the intensity of the light in case of plugging directly a LCD (Liquid Crystal Display), a connector for programming the microprocessor and three connectors so that the three I/O ports of the PSoC could be easily reached.

PSoC part *CY8C29466–24PXI* features [11]

Here are shown the most important features (not all of them) offered by this part according to the needs of this module of the project.

- Low power at high speed
- 3.0V (Volts) to 5.25V operating voltage
- 12 Rail-to-Rail analog PSoC blocks provide:
    - Up to 14-Bit ADCs
    - Programmable gain amplifiers
    - Programmable Filters and Comparators
- 16 Digital PSoC blocks provide:
    - 8- to 32-Bit timers, counters, and PWMs (Pulse-width modulator)
    - Multiple SPI masters or slaves
    - Connectable to all GPIOs Pins
- Internal ±2.5% 24/48 MHz oscillator
- 32K Bytes Flash Program Storage
- In-System Serial Programming (ISSP)
- Up to 12 analog inputs on GPIOs
- Pull up, pull down, high Z, strong, or open drain drive modes on all GPIOs
- Configurable interrupt on all GPIOs
- User-configurable low voltage detection
- On-chip precision voltage reference

The most important characteristics of this part are discussed in the next paragraphs as part of the PSoC configuration or in the interfacing with other modules.

Power supply

The remote control was aimed to be autonomous so there was no connection to an electricity point but batteries were opted for the supply. In a first step of the development of the remote control I was using lithium batteries as a power supply for this module, but since the consumption of it is quite low, the final decision was to use four AA 1.5V batteries in series, so that the voltage supply is 6 volts. The PSoC can both work with either 5 or 3.3 volts power supply. I finally chose working with 3.3 volts because the consumption of the system will be lower, which is very important for an autonomous design, and also because it makes easier the interface with the RF module. The power supply circuit, shown in figure 6, is based on the application notes of the LM1117, a low-dropout 3.3 volts regulator.

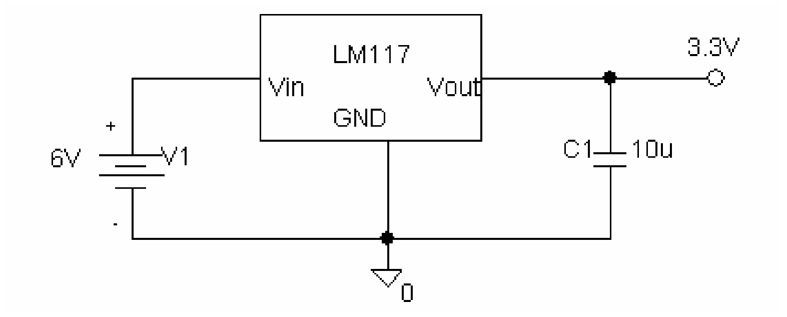

Figure 6. Power supply circuit of the remote control.

Speed and direction control circuit

The aim for this module is to create the signals needed for controlling the hovercraft but always trying to make the user interface as comfortable as possible.

In the first solution developed, this controlling was made by using simple buttons. The main advantage of this solution is that is quite comfortable for the user since it was very simple, only four buttons were needed (two of them for turning right or left, and the other two for changing the speed in both fans). Once in use the hovercraft it turned out that this solution was not good enough because you could not control how much the servo was turning (it was fixed) and you could only choose between two different speeds and that was not enough.

The next and final solution was made with potentiometers. The control signals that were taken from the middle pin of the potentiometers so that the circuit behaved as a controllable voltage divider. Two slide potentiometers were used to control the speed of the two fans and one rotary potentiometer to control the degrees that the servo turned. This solution improved the two main problems that the first one had because the user had more controllability on both the speed and the steering of the hovercraft and at the same time it was a simple and comfortable user interface. The final circuit for the remote control can be seen in the following figure. The power supply for this circuit was the 3.3 volts regulated signal.

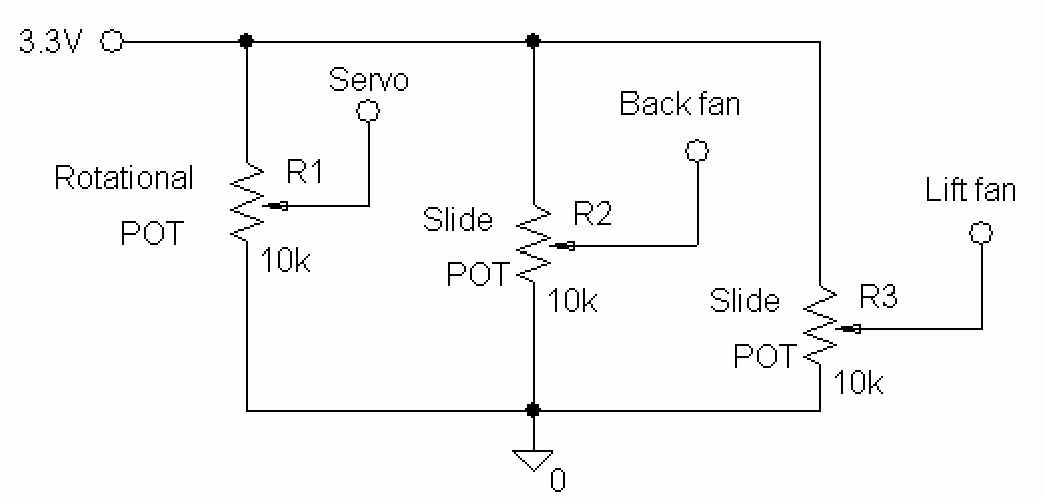

Figure 7. User interface of the remote control.

This circuit together with the power supply is mounted and soldered in a wire-wrapping board and connected to the PSoC board with wires.

RF transceiver module

The module used to perform the communications was the QFM-TRX1-24G transceiver module by Quasar based on the CC2500 chip and its application notes by Texas Instruments. The main features of the CC2500 chip are:

- Half duplex communications
- Easy interface with SPI
- Selectable frequency (2400 MHz – 2483 MHz)
- Programmable data rate from 1.2 to 500 kbps (kilobits per second)
- Low current consumption, 1.8 – 3.3 V operating voltage

In the remote control the transceiver works as a transmitter. The only problem posed by this module is that the pins are non-5V-tolerant so working with 3.3 volts avoids using two more ICs (Integrated Circuit) for adapting the logic levels. For connecting this module to an output port of the PSoC a small PCB has to be designed, the schematic is shown in next figure. It adapts the order of the signals coming from the port of the PSoC and going to the RF module. The board is plugged to a 10-pin connector of the *29466* board and the signals are put in the right order in the 8-pin output connector where the antenna is plugged. The layout and routing mask of the PCB are included in appendices 1 and 2, in figures 21 and 22 respectively.

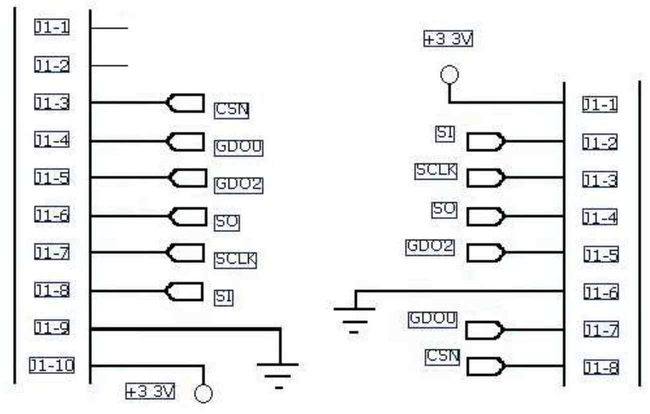

Figure 8. Schematic of the adapter for the antenna.

Configuration of the blocks in PSoC part *CY8C29466–24PXI*

The following list presents both the digital and the analog blocks needed for the remote control and their characteristics.

- SPI: it is needed to communicate the microprocessor with the RF module.
- LED (Light-emitting diode): there was no real need of a LED in the system but the SPI interface of the transceiver needed a chip select signal so a LED was opted to make it easier to drive it.
- PGA1 (Programmable gain amplifier): it amplifies the signal coming from the rotary potentiometer to work with the full scale of the ADC.
- PGA2: it amplifies the signal coming from one of the slide potentiometers to work with the full scale of the ADC.
- PGA3: it amplifies the signal coming from the other slide potentiometer to work with the full scale of the ADC.
- ADC: it converts the analog signal coming from PGA1 into a digital value of 8 bits.
- Dual ADC: it converts the two analog signals coming from PGA2 and PGA3 into digital signals of 8 bits.
- LCD: it is only used to help to debug the software.
- Pins 0, 1, 2 and 3 of port 0 are used to select the channel in which the RF module will be transmitting.

Finally, all these parts are assembled yielding the remote control. The schematic of the whole circuit is shown in appendix 3, in figure 19.

3.1.2 Hovercraft

The block diagram of the hovercraft is the one shown in figure 9. It consists of a communications block which is the input of information for the microprocessor and then three outputs controlled by PWM modules of PSoC to drive the servo and the fans.

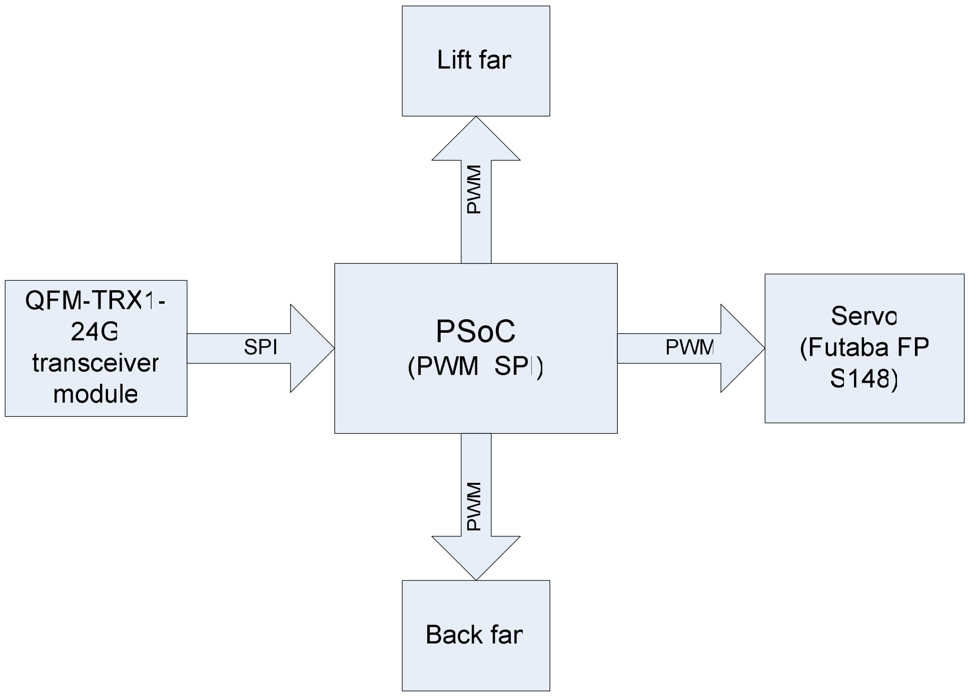

Figure 9. Block diagram of the hovercraft.

In this case the same decision must be taken about the PSoC part to be used. For the hovercraft not as many blocks as for the remote control were needed so a more simple part was adequate. PSoC part *CY8C27443–24PXI* was used since the university already had a PCB which included this PSoC part and I could use it to mount the prototype. This board contained a PSoC part *CY8C27443–24PXI*, a connector for the PSoC programmer, a reset button, a connector to interface with the RF module, external pins to be used with jumpers, a connector to plug a LCD, an input connector for the power supply and some other components that were not used.

PSoC *CY8C27443–24PXI* features [12]

Below are shown the most important features (not all of them) provided by this part to meet the needs of this module of the project.

- Low power at high speed
- 3.0V to 5.25V operating voltage
- 12 Rail-to-Rail analog PSoC blocks
- 8 Digital PSoC blocks provide:
  - 8- to 32-bit timers, counters, and PWMs
  - Multiple SPI masters or slaves
  - Connectable to all GPIOs Pins
- Internal ±2.5% 24/48 MHz oscillator
- 16K Bytes Flash Program Storage
- In-System Serial Programming (ISSP)
- Up to 12 analog inputs on GPIOs (General purpose I/O)
- Pull up, pull down, high Z, strong, or open drain drive modes on all GPIOs
- Configurable interrupt on all GPIOs
- User-configurable low voltage detection
- On-chip precision voltage reference

The most important characteristics of this part are discussed in the next paragraphs as part of the PSoC configuration or in the interfacing with other modules.

Power supply

The power supply for this part was more complex than the one used in the remote control. Firstly, I was also using Lithium batteries which supplied the power for the fans, the servo, and the PSoC. There were some problems with noise, caused by the power consumption of the two fans, so I decided to have two different power supplies, one for the fans and another one for the servo and the PSoC. For the fans I finally used four 3.2 volts Lithium phosphate rechargeable batteries which were more reliable and less dangerous since the previous ones could actually explode in case of wrong connection. Besides, I used four AA 1.5V batteries in series to supply both the servo and the PSoC. For the servo the voltage is not regulated since it supports up to 6V. The PSoC is run with 3.3V, this way the power consumption decreases and at the same time there is no need to adapt the signals going to the antenna. The following figure shows the power supply circuit.

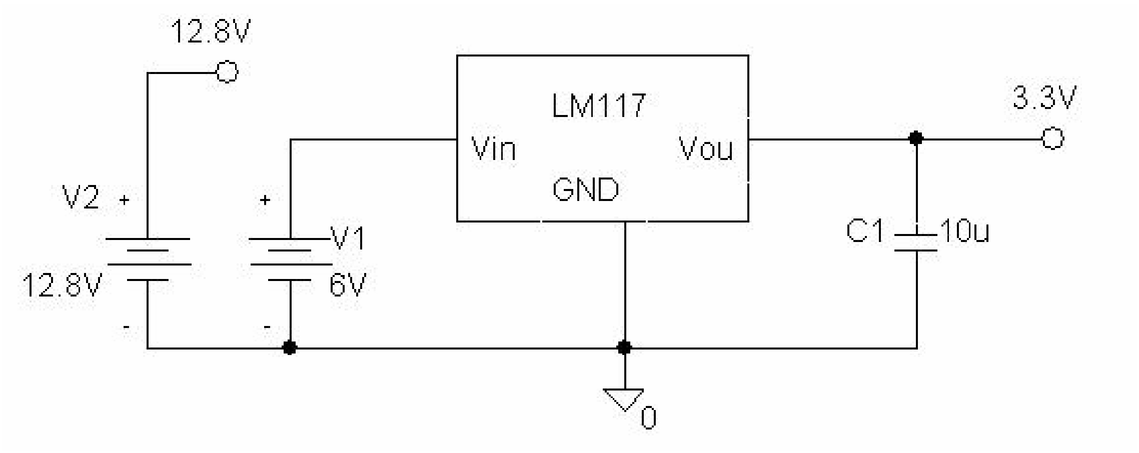

Figure 10. Power supply circuit of the hovercraft.

Servo

The steering of the hovercraft had to be controlled so that it turned in the way that the user indicated by using the remote control. This job is made by a servo. The servo I used to control the steering of the propeller fan was the S148 standard precision. It had the following characteristics:

- Motor: 3-pole
- Power requirement: 4.8V or 6V (from receiver)
- Power consumption: 6V and 8mA (at idle)
- Torque: 2.4 kg/cm at 4V or 3 kg/cm at 6V
- Transit Time: 0.28 sec/60° at 4.8V or 0.22 sec/60° at 6V
- Weight: 44.4g
- Mounting on-center dimensions
- Neutral pulse width control: 1.52ms

The servo consists of a small motor, a gear set, a feedback potentiometer, and some control electronics. In the figure below are shown the dimensions of the servo. The motor spins at variable speeds and is coupled to a reduction gear set that converts the motor's high speed into something that is more usable for its purpose. While this reduction is done torque (the 'twisting' power of the servo) is gained, and the more torque, the heavier the object the servo can move [13].

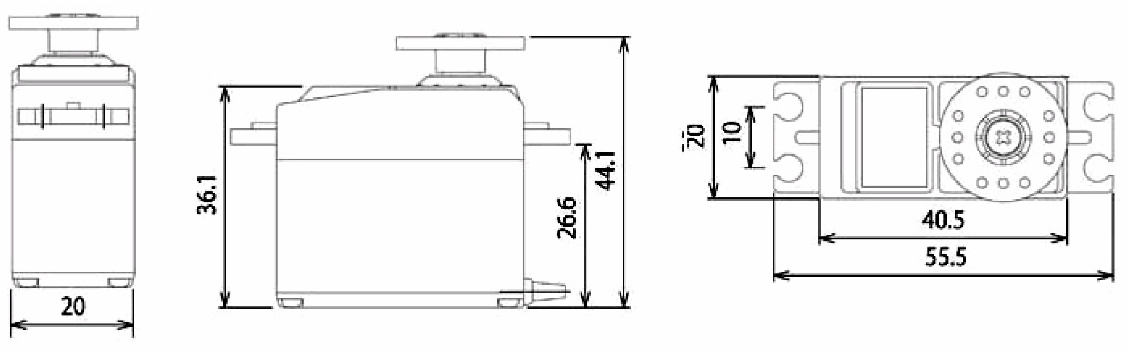

Figure 11. Dimensions of the servo S-148 [14].

A servo is a classic example of a closed-loop feedback system. The potentiometer is coupled to the output gear. Its resistance is proportional to the position of the servo's output shaft (0° to 180 °). This resistance is used by the control electronics to generate an error signal when the desired position is not the same as the current position. If a command is sent to a servo to place itself at 90° and the output shaft is actually at 45°, an error signal will cause the motor to move the head until the error signal is zero. The current position is 'fed-back' to the control system in a loop to maintain a zero error signal. The farther the actual position from the desired position, the faster the motor turns to bring the error back to zero.

The servo has 3 wires: power, ground and signal. The signal lead is used to send the positioning desired to the servo. The servo is controlled using a system called PCM (Pulse Code Modulation). The servo's electronics work in 20-milisecond blocks. For each block, the servo needs to see a positive-going pulse whose width in milliseconds tells it where to position the head (output shaft) from 0° to 180°.

The pulses needed to control a servo were generated by a PWM signal coming from the PSoC. As shown in the figure below, the middle signal (90º) had width of 1.52ms. The PWM signal had a frequency of 400Hz so it was sure that a going-positive pulse reached the servo every 20 ms.

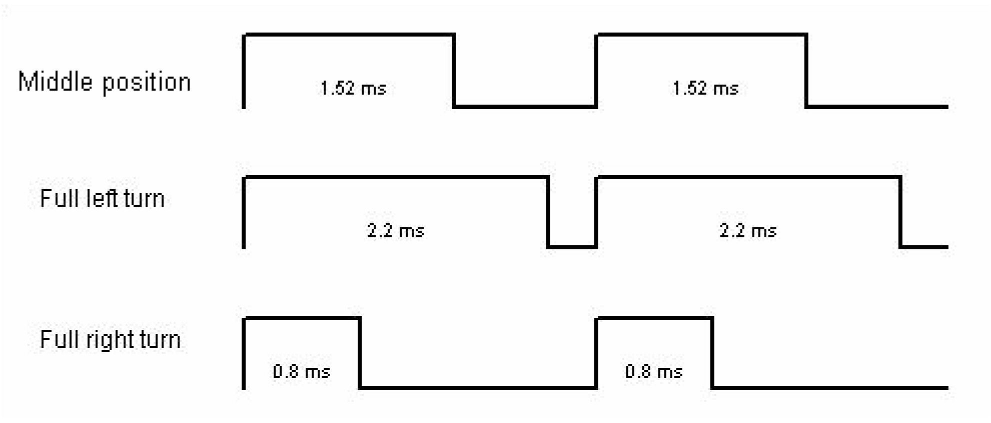

Figure 12. Examples of the signals that control the servo.

Fans

The hovercraft used two fans working as a propeller and as an impeller. The impeller is located in the middle of the chassis and glued to it. The propeller is mounted on the servo and glued to it so that both turn at the same time. The fans used are the EDF-64 as the impeller, and the EDF-55 as the propeller, both with motor EMI-150. The number of the fan referred to its diameter with the screws placed as shown in figure 13.

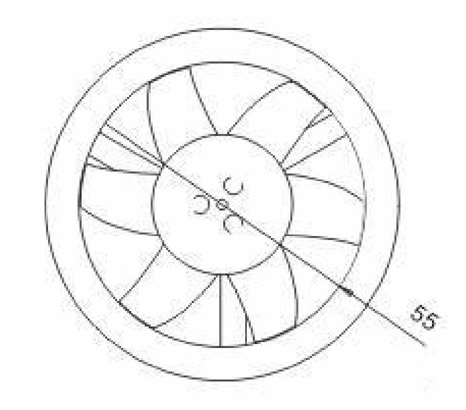

Figure 13. Diagram the fan EDF-55 [15].

These motors are precision machined to fit the EDF housing and combined with the screws give a very rigid base for the motor. Cooling holes allow the air to flow to the stator, cooling the windings so they can run more power. In the table below their characteristics can be seen:

	Weight (g)	**Volts (V)**	**Ampers (A)**	**Thrust (g)**	**Power (w)**	**Efficiency (g/w)**
EDF-55	59	10.8	6.6	152	71.28	2.13
EDF-64	67	10.8	8.3	201	89.64	2.24

Figure 14. Main characteristics of the fans [16].

The fans cannot be driven by a continuous signal from the batteries since they would burn so the solution is to drive them using two MOSFET (Metal Oxide Semiconductor Field Effect Transistor) transistors which are protected with zener diodes. One PWM signal of 100 kHz coming from the PSoC is applied to the gate of each transistor switching their operation mode so that the signal driving the fans is not continuous. Besides, a 10 kohms resistor is connected between the gate and the source of each transistor to avoid that the fans work without any signal in the gates. The figure below shows the position of the components in the circuit. Due to the high power of the signals going through the transistors, a heat sink is attached to each of them so that they do not burn.

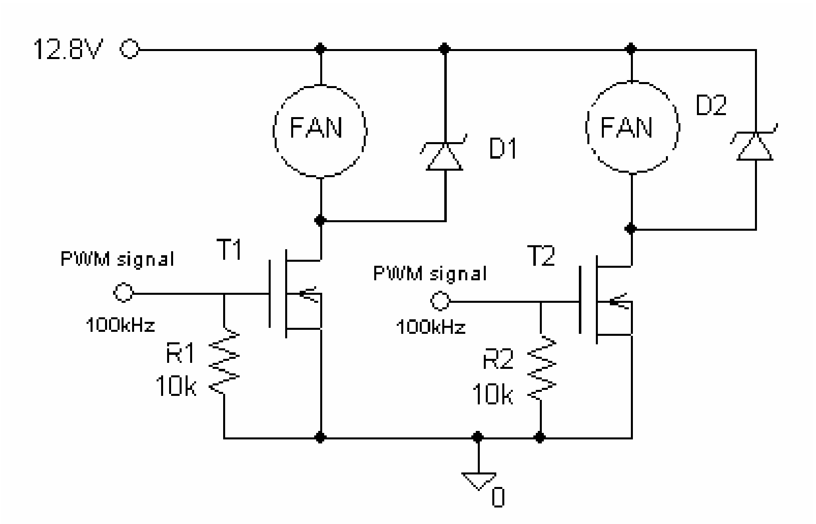

Figure 15. Circuit responsible of driving the fans.

RF transceiver module

The module used in the hovercraft was the same as used for the remote control, QFM-TRX1-24G transceiver, but this time it was working as a receiver. Since the PSoC wais run with 3.3 volts there was no need to adapt the signals. Besides, in the board I used the signals were already prepared so I did not need the PCB designed for the remote control.

Configuration of the blocks in PSoC part *CY8C29466–24PXI*

The following list enumerates both the digital and the analog blocks needed for the remote control and their characteristics.

- SPI: it is needed to communicate the microprocessor with the RF module.
- LED (Light-emitting diode): there was no real need of a LED in the system but the SPI interface of the transceiver needed a chip select signal so a LED was opted to make it easier to drive it.
- PWM16: this 16-bit PWM module is used to create the PCM signal with a frequency of 400Hz that controls the servo.
- PWM8_back: this 8-bit PWM is used to switch the working mode of the MOSFET transistor which is driving the propeller fan.
- PWM8_lift: this 8-bit PWM is used to switch the working mode of the MOSFET transistor which is driving the impeller fan.
- LCD: it was only used to help in debugging the software.
- Pins 4, 5, 6 and 7 of port 0 are used to select the channel in which the RF module will be transmitting.

Finally, these circuits were assembled yielding the electronics of the hovercraft. The schematic of the whole circuit is shown in appendix 3 figure 20.

3.2 Software

The source code of the whole system includes APIs for every analog or digital module used in PSoC but since they are explained in the datasheets of these modules only the code developed is commented. Besides, the whole source code for both the remote control and the hovercraft is shown in appendix 4.

3.2.1 Remote control

void ini()
This function performed the initialization of the remote control. It set the working mode of the SPI module and started it. It got the transmission channel selected by the user, initialized the RF module and set the channel in it. Besides, it initialized all the PGAs and the ADC modules preparing them for the first conversion.

BYTE PSoC_GetChannel()
It was called in the function above, it checked the pins 0, 1, 2, 3 of port0 to verify which channel had been selected for the RF communications.

void PSoC_BuildTxPacket(BYTE sequence_num, BYTE dest_addr, BYTE value)
This function received the data that was going to be sent and wrote it in the right location in the array that later on is transmitted.

void PSoC_SendTxPacket(BYTE resend_count, BOOL delay)
It was usually called right after the function above. It sent Tx_packet (the array built before), the number of times indicated by resend_count and performed a delay between them if required.

3.2.2 Hovercraft

void ini(void)

This function was very similar to the one used in the remote control. It also performed the initialization of the system, in this case, the receiver in the hovercraft, set the working mode of the SPI module and started it. It got the transmission channel selected by the user, initialized the RF module and set the channel in it. Furthermore, it initialized all the PWM modules disabling their interruptions and starting them.

BYTE PSoC_GetChannel(void)

It was called in the function above, it checked the pins 4, 5, 6, 7 of port0 to verify which channel had been selected for the RF communications.

void PSoC_HandlePacket(void)

This function checked if the reception of the array that was sent had been correct. Then it determined if the packet was different from the previous one and performed an additional check of the information. If everything was correct, it stored the data in the corresponding variables.

3.3 Manufacturing a PCB

The CAD software installed on the computers of Metropolia University was used for designing the PCBs. It consisted of: PADS Logic, PADS Layout and PADS Router. PADS Logic was used to define the schematics of the electronic circuit by selecting the components and making the electrical connections. PADS Layout was used to define the board outline, to place the footprints of the components in the board and to create the files needed to make the PCB. PADS Router was basically used to route the electrical connections on the board.

After designing the board with the CAD software the milling machine, shown in the next figure, was used to manufacture the PCB. This machine was controlled by software installed in the computer near it called BoardMaster. This programme required a list of commands consisting of bit information, coordinates and milling speed. For this purpose CAM-files of the design were generated using PADS layout, which contained a presentation of the PCB layout in ASCII (American Standard Code for Information Interchange) format. Then Circuit CAM had to be used to create the *.LM*D file needed by BoardMaster. First, the necessary CAM-files (acam.exc, acam.rep, acam.xgb, acam.xgo) from the previous step were imported. After, the cutout of the board and the breakout tabs on it were defined. Then, the rubuot layers were generated so that the program would know where the copper should be milled off. Finally, the *.LMD* file was generated and the PCB could be manufactured using BoardMaster [17].

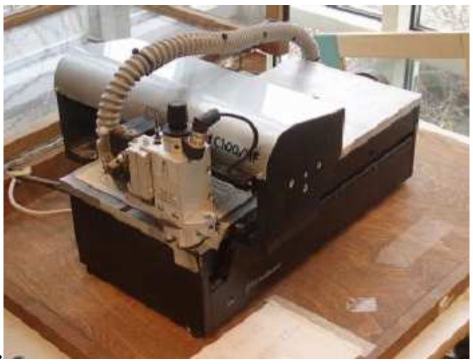
.

Figure 16. Milling machine.

3.4 The hovercraft

The hovercraft needed a chassis where all the hardware discussed before would be placed in order to make it work properly. This chassis was a rectangular cover, usually found in the boxes for packaging computers, made of expanded polystyrene beads. This material was very light so it was easy to lift it with a fan but on the other hand it was rigid enough to be used as a chassis. Two holes in this rectangular piece were made: the first one was a circular hole made in the middle of the chassis in order to place the impeller, the second one had the shape of the servo and it was placed close to the edge of the chassis as shown in the following figure:

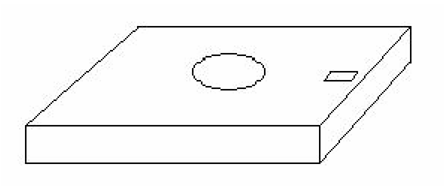

Figure 17. Diagram of the chassis.

Once the impeller and the servo were in place, both were fastened to the chassis with some adhesive tape. Then the propeller was glued to the servo. The power supply circuit was placed on the bottom part of the hovercraft and was also fixed to the chassis with adhesive tape. The PCB with the electronics and the MOSFET transistors which drive the fans were placed on the top part of the hovercraft, so that the transistors were cooled better, fixed with adhesive tape. The next figure shows the hovercraft and the remote control already mounted.

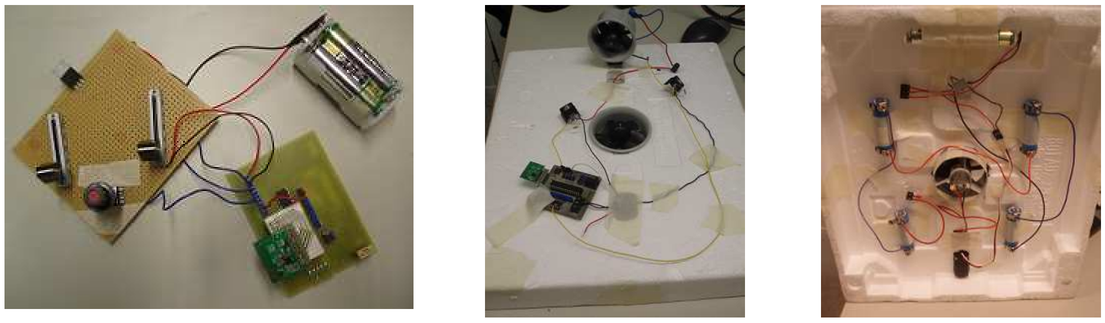

Figure 18. Remote control and hovercraft.

4 Discussion

Rather than manufacturing PCBs for each step of the development of the system insertion boards were used, as it was faster to build the circuits in this way. Besides, as usual in prototyping, the circuit changed several times as the process moved along so using PCBs would have been a waste of time and material.

The PSoC parts used both in the remote control and in the hovercraft have been chosen attending mainly to the needs of the system but also to the availability of developing boards in the university so that it was faster to start designing. For further developments, other parts could be needed in order to perform more complex tasks with this system.

Finally regular AA batteries were used for the power supply of the remote control since the power consumption expected is very low; however it is advisable to use rechargeable batteries if there is a continuous use of the system since it is always worthwhile economically.

The power supply in the hovercraft was divided into two parts to assure a continuous and smooth power supply for the PSoC. This task could have also be done using only the lithium batteries for every component in the system; nevertheless, the designer must take into account the noise generated by the high power consumption of the fans so that if only one power supply is used, a circuit that ensures and protects the supply of the PSoC part it is needed.

The batteries supplying the two fans ran out quite fast due to the high power consumption of the fans. In order to try to reduce this consumption the hovercraft could be mounted with a smaller chassis and lifted and propelled by low power fans so that at the same time fewer batteries are needed because in fact, the batteries are the heavier part of the hovercraft.

The potentiometers used in the remote control are all non-linear ones. They have been selected to make the user interface as comfortable as possible; besides using non-linear sliders for controlling both the lifting and the speed of the hovercraft is better because it is easier to prevent the user from selecting too low turning speed for the fans that could make them to burn. For steering control it is always easier to use a linear one, so only used the linear part of its rotational angle, from 210 to 270 degrees, was used. For further designs I is recommendable to use a linear one for the whole rotational angle so that the turning angle of the hovercraft is not limited, although turning the servo a lot makes the hovercraft very difficult to control.

5 Conclusions

The goal of this project was to design and manufacture a hovercraft as a platform for possible future development of more advanced tasks as part of a project suitable for any student at Metropolia.

The final result, a remote controlled model-size hovercraft, consists of two different parts, remote control and hovercraft that interact via RF antennas. The remote control permits the user to interact with the hovercraft to control the lifting, the speed and the steering of it. Both parts have been developed around PSoC controllers which perform the needed functionality in each case.

The final performance of the system is not as accurate as desired. While the user interface is very friendly and intuitive, the control of the self drifting of the hovercraft is no as good as it was expected. For further developments on the platform the lack of friction resistance might be avoided with a plastic skirt inflated by the lifting fan or with inflated plastic attached to the edges of the hovercraft.

Finally, for further projects using this platform two possibilities are suggested: developing software and hardware necessary so that the remote control can be performed from a computer, or making a remote control in which the user can specify the desired heading of the hovercraft and do the control of the orientation with electromagnetic devices.

References

1 Barr, M. Embedded Systems Glossary [online]. Netrino Technical Library.
URL: http://www.netrino.com/Embedded-Systems/Glossary.
Accessed 12th March 2009.

2 Balarin, F. Hardware-software co-design of embedded systems: the POLIS approach. Primary Material: Book Published: Boston: Kluwer Academic, 1997.

3 Alarm:clock euro site [online].
URL: http://www.thealarmclock.com/euro
Accessed 12th March 2009.

4 Tikkanen H. Printed circuit board design guide: using modern CAD systems, examples from PADS. Jyväskylä, Finland : DS-Design Systems, 2004.

5 Gizmo highway site. The history of the hovercraft [online].
URL: http://www.gizmohighway.com/history/hovercraft.htm
Accessed 22nd March 2009.

6 Technology and information links999 website. The history of the hovercraft [online].
URL: http://www.links999.net/hovercraft/hovercraft_history.html
Accessed 22nd March 2009.

7 Technology and information links999 website. Military hovercraft [online].
URL: http://www.links999.net/hovercraft/mem_hov/hovercraft_military.html
Accessed 22nd March 2009.

8 Datasheet [online]. Cypress Semiconductor Corporation; 2004-2008.
URL: http://www.cypress.com/?rID=3337
Accessed 16th February 2009.

9 Serial buses information page in ePanorama website [online].
URL: http://www.epanorama.net/links/serialbus.html
Accessed 18th March 2009.

10 Stallings, W. Data and Computer Communications (6th ed.). New Jersey, USA: Prentice Hall, 2000.

11 Datasheet [online]. Cypress Semiconductor Corporation; 2004-2008
URL: http://www.cypress.com/?rID=3334
Accessed 18th February 2009.

12 Datasheet [online]. Cypress Semiconductor Corporation; 2004-2008.
URL: http://www.cypress.com/?rID=3314
Accessed 26th February 2009.

13 Application note #106 version 1.0. HVW Technologies Inc. [online].
URL: http://www.HVWTech.com
Accessed 4th March 2009.

14 Specifications of the product of ServoCity [online].
URL: http://www.servocity.com/html/s148_standard_precision.html
Accessed 3rd April 2009.

15 Datasheet [online]. Grand wing servo technology co., ltd.
URL: http://www.gws.com.tw/english/product/powersystem/edf55.htm
Accessed 9th March 2009.

16 Datasheet [online]. Grand wing servo technology co., ltd.
URL: http://www.gws.com.tw/english/product/powersystem/edf64.htm
Accessed 9th March 2009.

17 Piironen, A. Printed circuit board fast design and manufacturing guide. Espoo, Finland: Metropolia University of Applied Sciences.

Appendices

Appendix 1: PCB Layout

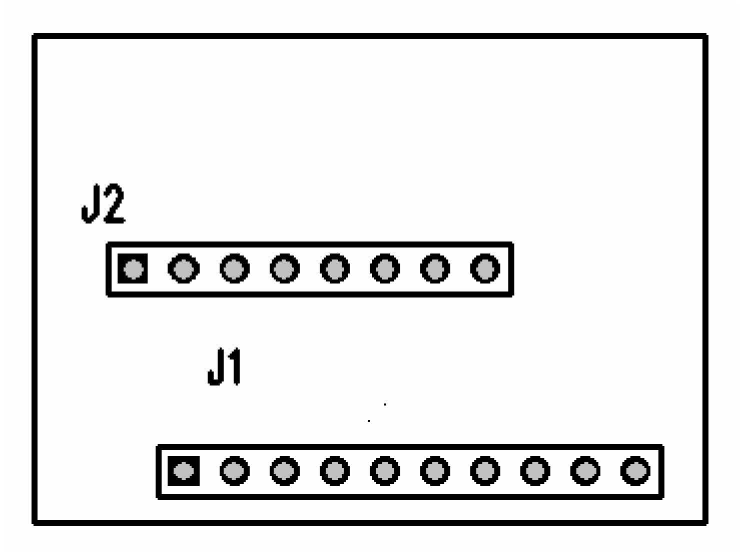

Figure 21. Layout of the adapter for the RF module.

Appendix 2: PCB Routing

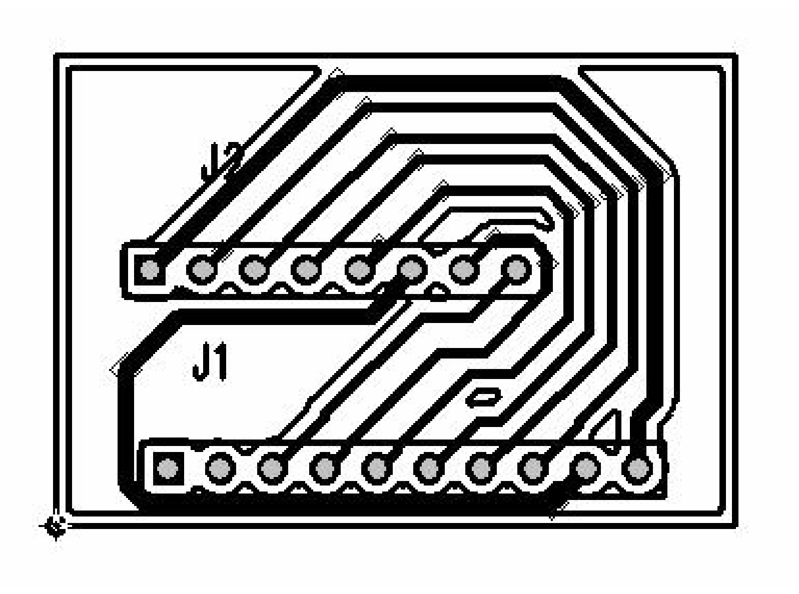

Figure 22. Routing mask of the adapter for the RF module.

Appendix 3: Schematics

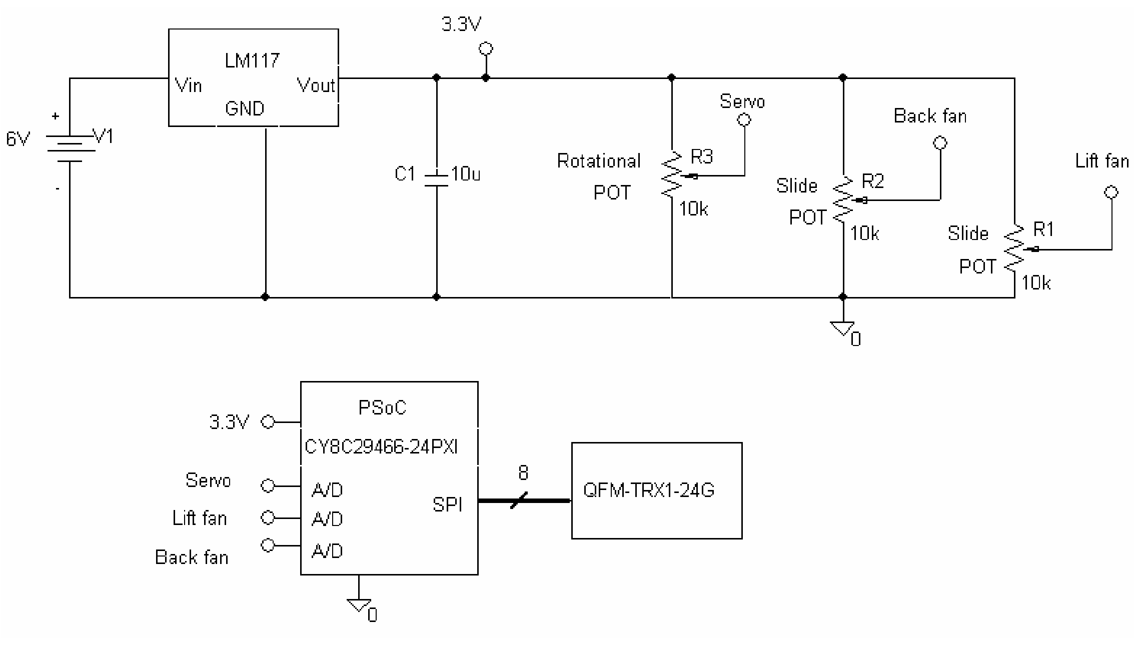

Figure 19. Circuit of the remote control.

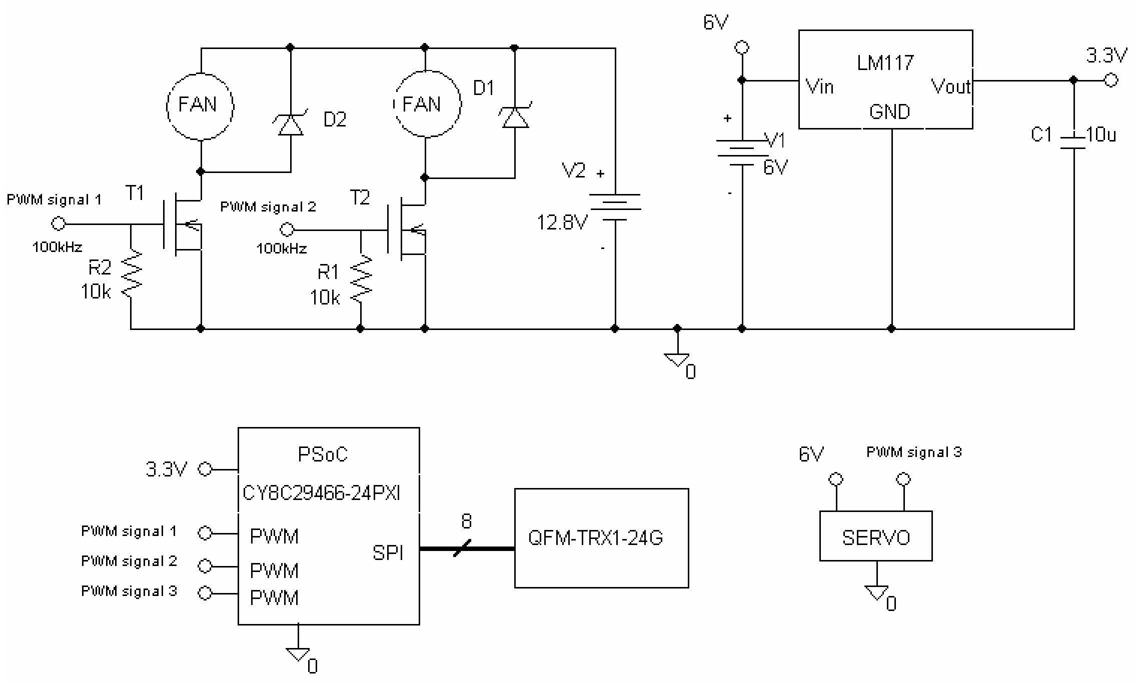

Figure 20. Circuit of the hovercraft.

Appendix 4: Source code

Remote control

```
//----------------------------------------------------------------------------
// C main line
//----------------------------------------------------------------------------

#include <m8c.h>            // Part specific constants and macros
#include "PSoCAPI.h"        // PSoC API definitions for all User Modules
#include "PSoCGPIOINT.h"    // General purpose I/O
#include "rf.h"             // The RF module API


/* Function prototypes */
void ini(void);
BYTE PSoC_GetChannel(void);
PSoC_BuildTxPacket(BYTE sequence_num,BYTE dest_addr,BYTE value);
void PSoC_SendTxPacket(BYTE resend_count, BOOL delay);

#define DEBUG

/* Tx_packet legend */
#define LENGTH 0
#define DEST_ADDR 1
#define SEQUENCE_NUM 2
#define VALUE 3
#define INV_VALUE 4

/* Address legend*/
#define BACK_FAN 0x10
#define LIFT_FAN 0x20
#define SERVO 0x30

/* End of legend */
#define PACKET_SIZE 5                   // Size of Tx_packet
#define CHANNEL_BIT_MASK 0x0F           // Used for the jumpers
#define RESEND_RATE 2
#define DELAY TRUE
#define NO_DELAY FALSE
#define DELAY_CONSTANT 500
```

Appendix 4: Source code

```
/* Global variables */
BYTE Tx_packet[PACKET_SIZE] = {0, 0, 0, 0, 0};
unsigned int Sequence_num = 0;
BYTE RF_channel = 0;

void main()
{
        char dir = 130;         //middle position of the servo
        char speed = 0, lift = 0;

        ini();

        while(1)
        {
                if(DIR_fIsDataAvailable())              //Data of the servo
                {
                        dir = DIR_cGetDataClearFlag();
                        LCD_Position(0,0);
                        LCD_PrHexByte(dir);
                        if(dir < 6)                     //Limit for linear working
                                dir = 6;
                        else
                        {
                                if(dir > 42)
                                        dir = 42;
                        }

                        PSoC_BuildTxPacket(++Sequence_num, SERVO, dir);
                        PSoC_SendTxPacket(RESEND_RATE, NO_DELAY);

                        #ifdef DEBUG
                        LCD_Position(1,3);
                        LCD_PrHexByte(Tx_packet[LENGTH]);
                        LCD_Position(1,5);
                        LCD_PrHexByte(Tx_packet[DEST_ADDR]);
                        LCD_Position(1,7);
                        LCD_PrHexByte(Tx_packet[SEQUENCE_NUM]);
                        LCD_Position(1,9);
                        LCD_PrHexByte(Tx_packet[VALUE]);
                        LCD_Position(1,11);
                        LCD_PrHexByte(Tx_packet[INV_VALUE]);
                        #endif
                }
```

Appendix 4: Source code

```
		if(DUALADC_fIsDataAvailable())	//Data for the fans
		{
			speed = DUALADC_iGetData1();
			lift = DUALADC_iGetData2ClearFlag();

			if(speed > 60)			//Maximum speed
				speed = 60;

			PSoC_BuildTxPacket(++Sequence_num, BACK_FAN, speed);
			PSoC_SendTxPacket(RESEND_RATE, NO_DELAY);

			if(lift > 60)			//Maximum lift
				lift = 60;

			PSoC_BuildTxPacket(++Sequence_num, LIFT_FAN, lift);
			PSoC_SendTxPacket(RESEND_RATE, NO_DELAY);

			#ifdef DEBUG
			LCD_Position(0,9);
			LCD_PrHexByte(speed);
			LCD_Position(0,6);
			LCD_PrHexByte(lift);
			#endif

			DUALADC_GetSamples(1);
		}
	}
}

void ini(void)
{
	M8C_EnableGInt;			//Enable global interrupts

	LCD_Start();


	/* Initialize SPIM and the RF module itself */
	SPIM_Start(SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST);
	SPIM_CS_Start();
	SPIM_CS_On();

	/* Obtain channel*/
	RF_channel = PSoC_GetChannel();
```

Appendix 4: Source code

```
        /* The RF module */
        rf_init();
        rf_setChannel(RF_channel);

        /* ADC for the servo*/
        PGA_DIR_Start(PGA_DIR_HIGHPOWER);
        DIR_Start(DIR_HIGHPOWER);
        //DIR_WritePulseWidth(0);
        DIR_StartAD();

        /* DUALADC for the fans*/
        PGA_LIFT_Start(PGA_LIFT_HIGHPOWER);
        PGA_BACK_Start(PGA_BACK_HIGHPOWER);
        DUALADC_Start(DUALADC_HIGHPOWER);
        DUALADC_GetSamples(1);          //This way it will get only one sample

        #ifdef DEBUG
        LCD_Position(1,14);
        LCD_PrHexByte(RF_channel);
        #endif
}

BYTE PSoC_GetChannel(void)
{
        return C0_Data_ADDR & CHANNEL_BIT_MASK;
}


void PSoC_BuildTxPacket(BYTE sequence_num, BYTE dest_addr, BYTE value)
{
        Tx_packet[LENGTH] = PACKET_SIZE;
        Tx_packet[DEST_ADDR] = dest_addr;
        Tx_packet[SEQUENCE_NUM] = sequence_num;
        Tx_packet[VALUE] = value;
        Tx_packet[INV_VALUE] = ~value;
}
```

Appendix 4: Source code

```
void PSoC_SendTxPacket(BYTE resend_count, BOOL delay)
{
        unsigned int i = 0, j = 0;

        for(i = 0; i < resend_count; i++)
        {
                rf_sendPacket(Tx_packet,PACKET_SIZE);

                if(delay)
                {
                        for (j = 0; j < (DELAY_CONSTANT); j++)
                        {
                                asm("nop");
                        }
                }
        }
}
```

Hovercraft

```
//------------------------------------------------------------------------
// C main line
//------------------------------------------------------------------------

#include <m8c.h>              // part specific constants and macros
#include "PSoCAPI.h"          // PSoC API definitions for all User Modules
#include "PSoCGPIOINT.h"  // useful pin definitions
#include "rf.h"

/* Function prototypes */
void ini(void);
BYTE PSoC_GetChannel(void);
void PSoC_HandlePacket(void);       // Receives and handles a packet

#define DEBUG

/* Rx_packet legend */
#define LENGTH 0
#define DEST_ADDR 1
#define SEQUENCE_NUM 2
#define VALUE 3
#define INV_VALUE 4
```

Appendix 4: Source code

```
/* Address legend*/
#define BACK_FAN 0x10
#define LIFT_FAN 0x20
#define SERVO 0x30

/* end of legend */
#define PACKET_SIZE 5
#define RF_WAIT 1
#define RF_NO_WAIT 0
#define RF_BUFFER_SIZE 61         // Packet length is 61 bytes, though we use less
#define REPORT_OUT_SIZE 8
#define DATA_BUFFER_SIZE 65
#define DATA_BUFFER_EMPTY 0
#define KEYSTROKE_SCALE_FACTOR 1
#define CHANNEL_BIT_MASK 0xF0
#define CHANNEL_BIT_SHIFT_MASK 0x08
#define CHANNEL_BIT_SHIFT 4


/* Global variables */
BYTE Rx_buffer[RF_BUFFER_SIZE];
BYTE RF_channel = 0;
BYTE address = 0, sequence_num = 0, value = 0;              //Data received
BOOL new_packet = FALSE;


void main()
{
        char error [] = "ERROR";
        int pulse = 0;
        ini();

        while(1)
        {
                PSoC_HandlePacket();
                if(new_packet)
                {
                        switch(address)                 //Determine type of package

                        {
                                case SERVO:
```

Appendix 4: Source code

```
                #ifdef DEBUG
                LCD_Position(0,0);
                LCD_PrHexByte(value);
                #endif

                //Convertion to pulse-width for the PWM
                pulse = 394 + ((42 - value)*20)/3;
                PWM16_WritePulseWidth(pulse);

        break;


        case BACK_FAN:
                if(value < 4)  //To avoid slow working of the fan
                        value = 0;
                else
                        //Convertion to pulse-width for the PWM
                        value = (value/2)+20;

                #ifdef DEBUG
                LCD_Position(0,6);
                LCD_PrHexByte(value);
                #endif

                BACK_FAN_WritePulseWidth(value);

        break;
        case LIFT_FAN:
                if(value > 9)   //To avoid slow working of the fan
                        //Convertion to pulse-width for the PWM
                        value = (value/2)+20;
                else
                        value = 0;

                #ifdef DEBUG
                LCD_Position(0,9);
                LCD_PrHexByte(value);
                #endif

                PWM8_WritePulseWidth(value);

        break;
```

Appendix 4: Source code

```
                    default:
                        #ifdef DEBUG
                        LCD_Position(0,0);
                        LCD_PrString(error);
                        #endif
                    break;
                }
                new_packet = FALSE;
            }
        }
}

void ini(void)
{
    #ifdef DEBUG
    LCD_Start();
    #endif

    /* Initialize SPIM and the RF module itself */
    SPIM_CS_Start();
    SPIM_CS_On();
    SPIM_Start(SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST);

    RF_channel = PSoC_GetChannel();

    #ifdef DEBUG
    LCD_Position(1,14);
    LCD_PrHexByte(RF_channel);
    #endif

    rf_init();
    rf_setChannel(RF_channel);

    PWM16_DisableInt();         //Disable PWM16 interrupt
    PWM16_Start();

    BACK_FAN_DisableInt();
    BACK_FAN_Start();

    PWM8_DisableInt();     //Disable PWM8 interrupt
    PWM8_Start();

}
```

Appendix 4: Source code

```
BYTE PSoC_GetChannel(void)
{
	int i = 0;
	BYTE reading = 0x00;
	BYTE channel = 0x00;
	BYTE mask = CHANNEL_BIT_SHIFT_MASK;

	reading = C0_Data_ADDR & CHANNEL_BIT_MASK;

	/* Swap the bits, due to pcb fault */
	for (i = 0; i < CHANNEL_BIT_SHIFT; i++)
	{
		reading>>=1;
		if (reading & CHANNEL_BIT_SHIFT_MASK)
		{
			channel |= mask;
		}
		mask>>=1;
	}
	return channel;
}


void PSoC_HandlePacket(void)
{
	BOOL crc_ok = FALSE;
	unsigned char packet_length = RF_BUFFER_SIZE, rssi = 0, lqi = 0;
	//getRxPacket variables

	crc_ok = rf_getRxPacket(Rx_buffer, &packet_length, &rssi, &lqi, RF_WAIT);

	if(crc_ok)
	{
		#ifdef DEBUG
		LCD_Position(1,3);
		LCD_PrHexByte(Rx_buffer[LENGTH]);
		LCD_Position(1,5);
		LCD_PrHexByte(Rx_buffer[DEST_ADDR]);
		LCD_Position(1,7);
		LCD_PrHexByte(Rx_buffer[SEQUENCE_NUM]);
		#endif
```

Appendix 4: Source code

```
		if(sequence_num != Rx_buffer[SEQUENCE_NUM])
		{
			#ifdef DEBUG
			LCD_Position(1,9);
			LCD_PrHexByte(Rx_buffer[VALUE]);
			LCD_Position(1,11);
			LCD_PrHexByte(Rx_buffer[INV_VALUE]);
			LCD_Position(0,14);
			#endif

			/* Additional check */
			if (Rx_buffer[VALUE] == ~Rx_buffer[INV_VALUE])
			{
				#ifdef DEBUG
				LCD_PrHexByte(34);
				#endif
				/* Save info and buffer data */
				sequence_num = Rx_buffer[SEQUENCE_NUM];
				value = Rx_buffer[VALUE];
				address = Rx_buffer[DEST_ADDR];
				new_packet = TRUE;
			}
			else
			{
				#ifdef DEBUG
				LCD_PrHexByte(17);
				#endif
			}
		}
	}
	else
	{
		#ifdef DEBUG
		LCD_Position(1,3);
		LCD_PrHexByte(255);
		#endif
	}
}
```