Universal remote control for Biorob's robots

Matthieu Tardivon

Semester project at the Biorobotics laboratory
Ecole Polytechnique Federale de Lausanne

Professor: Auke Jan Ijspeert
Assistant: Alessandro Crespi

Autumn semester 2012

Abstract

This semester project aims at the development and realization of a new version of the autonomous remote control for robots that was made by G. Cuendet (former student) three years ago. The new remote control must be able to control the Salamandra robotica and the Amphibot robots, must be more ergonomic, cheaper, and present some new features, such as the possibility to flash a robot using a firmware contained in a SD card on the remote control. After a quick review of the past work, a first prototype was realized in order to develop the software. Then, a SolidWorks model of the remote control was made to build it using a 3D printer. Finally, a final PCB was made to fit into the controller.

Contents

Chapter 1

Introduction

1.1 Gabriel Cuendet's remote control

Before Gabriel Cuendet's semester project (3 years ago), a computer was needed in order to remotely control an AmphiBot or Salamandra robotica robot. Thanks to Gabriel Cuendet, it is now possible to control these robots with an autonomous remote control. This remote control contains a LCD screen, 4 buttons, a joystick and a radio interface. Even if this seems enough to control a robot, several problems appeared while using this remote control.

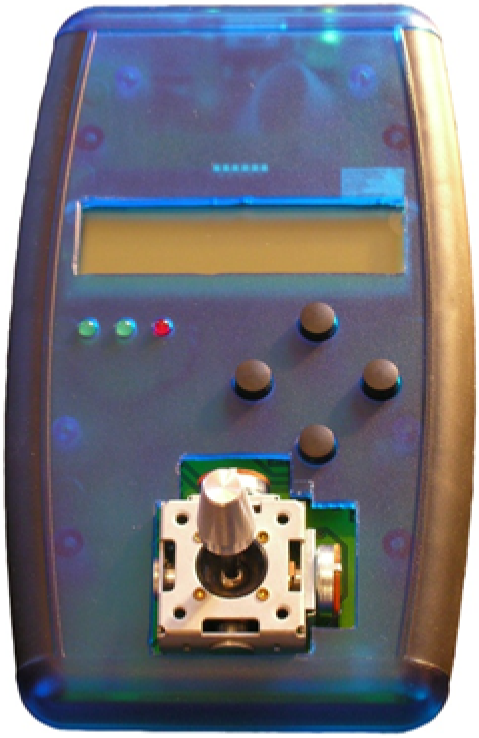


Figure 1.1: G. Cuendet's remote control (Source: G.Cuendet's project presentation)

First of all, the box is just large enough to be held in one hand and you need the other to move the joystick as it too high for the thumb which holds the remote control. This problem of ergonomics can make users who are not familiar to this remote control quite uncomfortable while piloting a robot. Moreover, the joystick is industrial, very expensive (around CHF 100) and not waterproof, which can be a problem when controlling amphibious robots (water splashing).

However, the main problem remains universality. The code made by Gabriel Cuendet implements one function per kind of robots in the Biorob lab as they work differently. But if a new robot is developed by the lab, there will be a new software to be developed for the remote control. There will be the same problem if the code of one of the robots is modified (e.g. add of a new mode). Therefore, there is a need to create a more ergonomic universal remote control with some new features. Though this remote control aims at only controlling Salamandra robotica and Amphibot robots, zigbee communication has been added in order to being able to control more robots In the future.

Among the new features, the most interesting is the possibility of flashing a robot with the remote control, using a SD Card. Indeed, the user will just have to bring a SD Card with the latest firmware with him for a presentation, which is lighter than a computer. Another typical application would be replacing a firmware used for experiments (but not good for demos) with a "standard" one that works.

1.2 Project description

Universal remote control for Biorob's robots

Semester project, project description

Matthieu Tardivon, October 1st

The project aims at the development of a new wireless remote control to substitute for the one made by G. Cuendet three years ago. The remote control will have to be able to communicate with at least with Salamandra robotica and Amphibot robots, and to flash them thanks to an SD card on board. As they have already been designed, the electronics for the RF and the voltage supply part are not part of the project. The ergonomics will also be enhanced.

Task list:

1. Review of the project made by G. Cuendet in order to integrate the RF and voltage supply electronics on the PCB.
2. Listing of the functionalities of the remote control in order to determine all the components needed.
3. Microcontroller pin mapping and hardware design (PCB).
4. Development of the software on the microcontroller.
5. Mechanical design of the remote control in order to improve the ergonomics.

Deliverables:

1. The present report.
2. The mounted, working remote control.
3. An entry on the Biorob website
4. A CD with all the project materials (source code, PDF of report, SolidWorks model, etc...).

Chapter 2

Hardware design

During this project, a review of the remote control's schematics made by G. Cuendet enabled the integration of the radio interface (designed originally by Alessandro Crespi), the battery management and power supply in the new remote control. Though the project did not focus on these electronic parts, two small sections will explain their functioning.

The electronic part of the project mainly focused on what will be called the main interface (microcontroller and all its peripherals). A whole section will detail and explain the choices made for this interface.

Two PCBs were made during the project. The first one was a prototype in order to allow the development of the software and the second one was the final version, which fits into the case made in the mechanical design phase of the project.

All the schematics and 3D model of each PCB are available at the end of the report.

2.1 Radio interface

The radio interface designed by Alessandro Crespi can be seen as a block communicating with the microcontroller of the main interface through a UART connection.

This block is controlled by a Microchip PIC16LF876A microcontroller which communicates through a SPI bus with a Nordic nRF905 868MHz radio transceiver.

The software running on the PIC16LF876A was also made by Alessandro Crespi and was thus not part of the project.

An important detail about the routing of this interface on the PCB is that the tracks must be as short as possible between the nRF905 and the antenna connector. Moreover, the track coming from the center of the antenna connector is critical and must be the shortest as possible and must have a specific width in order to have a 50 Ohm impedance (1.5mm width).

2.2 The battery management and voltage supply

The battery management and power supply electronics have been integrated to the schematics from the ones made by G. Cuendet.

In order to function, the remote control uses a rechargeable Li-Ion battery which needs thus to be protected. The possibility of charging the battery by plugging an external power source is also included in this interface.

The board only needs a supply of 3.3V and as the battery voltage range is between 4.2V and 3V according to its charge, there was a need of using a DC/DC converter to provide constant voltage. It is the Linear Technology LTC3240-3.3 which makes this conversion.

To monitor the parameters of the battery (voltage, current, temperature) and protect it from dangerous conditions (overvoltage, undercharging, short circuit, etc...), a Dallas Semiconductor DS2764 is used and communicates with the microcontroller of the main interface through an I2C bus.

Finally, another component from Linear Technology, the LT1571-5 is used to handle the battery charging. The charge voltage can be between 8.2V and 26V, and the chip is connected to a power jack connector.

A possibility for further development would be to make the battery also rechargeable through a microUSB connector. As almost everybody has a Smartphone and often its charger nearby, it would be a good solution in case of the usual big charger was forgotten or broken.

2.3 The main interface

The main interface is one composed by the main microcontroller and its peripherals. This interface is the most important part of the remote control as it communicates with the radio interface, the battery management and most important of all, the user.

2.3.1 Choice of the microcontroller

While G.Cuendet used a Microchip PIC18F2580 as a microcontroller for its remote control, another was chosen for this project. Indeed, there was a need for a microcontroller with more memory and speed so that it can read a file system on a SD card for instance.

As I had already worked with it and met several enterprises (Melexis, Schlumberger, eSmart...) using STM32 microcontrollers (STMicroelectronics), a STM32F103CBT6 was chosen for this project. This microcontroller works at a 72MHz with an ARM Cortex M3, 128kB of FLASH, 20kB of internal RAM and only 48 pins. All of the 48 pins have been used in the main interface.

Another advantage of the STM32 microcontrollers is that they are quite cheap and have reduced power consumption. As this is the first time this microcontroller is used in the Biorob lab, this project will also serve as a test for possible future uses of this microcontroller in other projects.

2.3.2 Functionalities

The remote control must have the following features:

- Communication at least with Salamandra robotica and Amphibot robots, if possible with Zigbee controlled robots (e.g. Pleurobot)
- Control of the robots
- Display of a menu (user-friendly)
- Remote flashing of robots supporting this function

2.3.2.1 Communication

As this remote control must be as universal as possible, two ways of communicating with a robot have been implemented in the hardware design. The first one is the radio communication mentioned in the previous dedicated section and the second is the zigbee communication a Digi Xbee S2 module.

As two types of Xbee modules exist, Xbee S2 ZB RF and Xbee 802.15.4 RF, which do not have the same networking connection protocol at all, the possibility of unplugging and replacing the device for the other will be given if the mechanical integration allows it. Both of them communicate using UART with the microcontroller.

2.3.2.2 Robot control

In the previous version of the remote control, only one industrial and expensive joystick was used to control the robots. The microcontroller has many pins and future Biorob robots might require more than one joystick to pilot them. Therefore, two Sony Playstation like, cheap (less than CHF 5) joysticks were integrated on the board. Moreover, even if one joystick is enough to control a robot (one axis: the direction, the other: the speed), some users might prefer to use two (one joystick for the direction, the other for the speed).

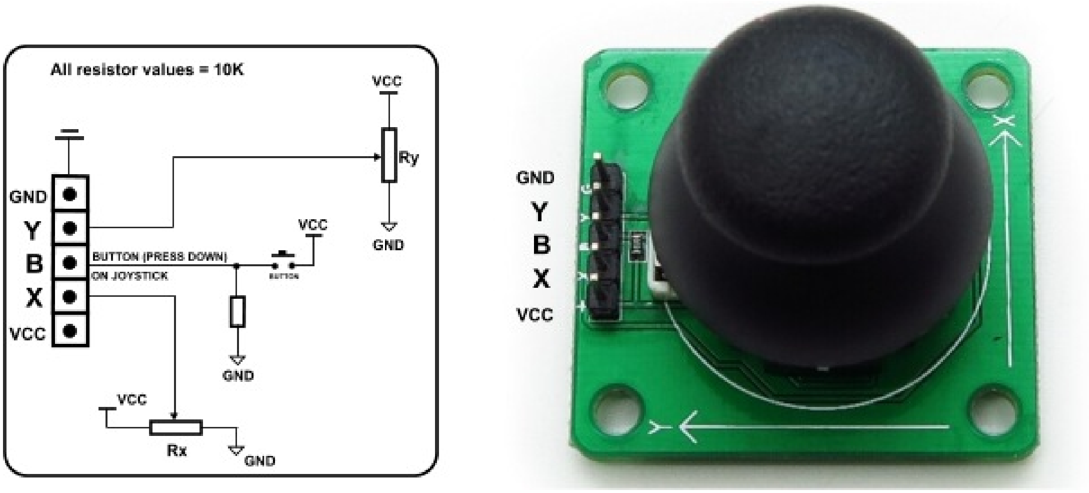


Figure 2.1: Joysticks used in the remote control with its schematics (Source: http://robotics.org.za/)

Each axis of the joysticks has a potentiometer and requires just an ADC on the microcontroller to get its value. The STM32 returns a value between 0 and 4095, which is quite a large range and gives an interesting precision on the joystick.

Finally, each joystick can be pressed as a button, which gives more possibilities, such as changing mode (e.g. walking/swimming for the salamander).

2.3.2.3 User interface

In order to provide the user with a pleasant interface, the remote control has a LCD screen and four push buttons. The LCD screen used is the same than G.

Cuendet (Batron BTHQ21605V-FSRE-I2C-COG) which has the great advantage of communicating through an I2C bus (minimization of the number of pins used) and a voltage supply of only 3.3V (contrary to many others which need a voltage supply of 5V). This screen displays two lines of 16 alphanumeric characters maximum, which is quite enough to display a menu.

Besides this visual interface, the remote control can be controlled using four buttons, positioned on a cross, in the Sony Playstation like way. The top and bottom button can be used to navigate up and down through the current menu. The left button can be used to get to the previous menu and the right button validates the current choice.

2.3.2.4 Remote flashing

The flashing of a robot requires the remote control to read a file. Therefore, a SD card slot was added in the remote control. It communicates through a SPI bus with the STM32 microcontroller. This feature gives the possibility of having many versions of firmware of different robots on the remote control. Thus, it will be able to flash any robot of the lab supporting this function, in a convenient way.

2.3.2.5 Other features

In order to flash the STM32F103CBT6, a JTAG probe is connected on a JTAG connector (header 2x10 pins) on the board. The JTAG interface allows the developer to know the exact state of the processor at any moment, which can be very useful while debugging.

Three other ways of debugging have been added on the board: a serial port, a RGB LED and a buzzer (it can also play music while piloting a robot).

For further development, a micromatch connected to the I2C bus has been added if one would like to add an IMU (e.g. FreeIMU - viacopter.eu) and control a robot with a Nintendo WiiMote like remote control.

2.4 Overview of the board

The complete schematics can be found at the end of the report but the *little* figure below should help the reader to understand the rough functioning of the remote.

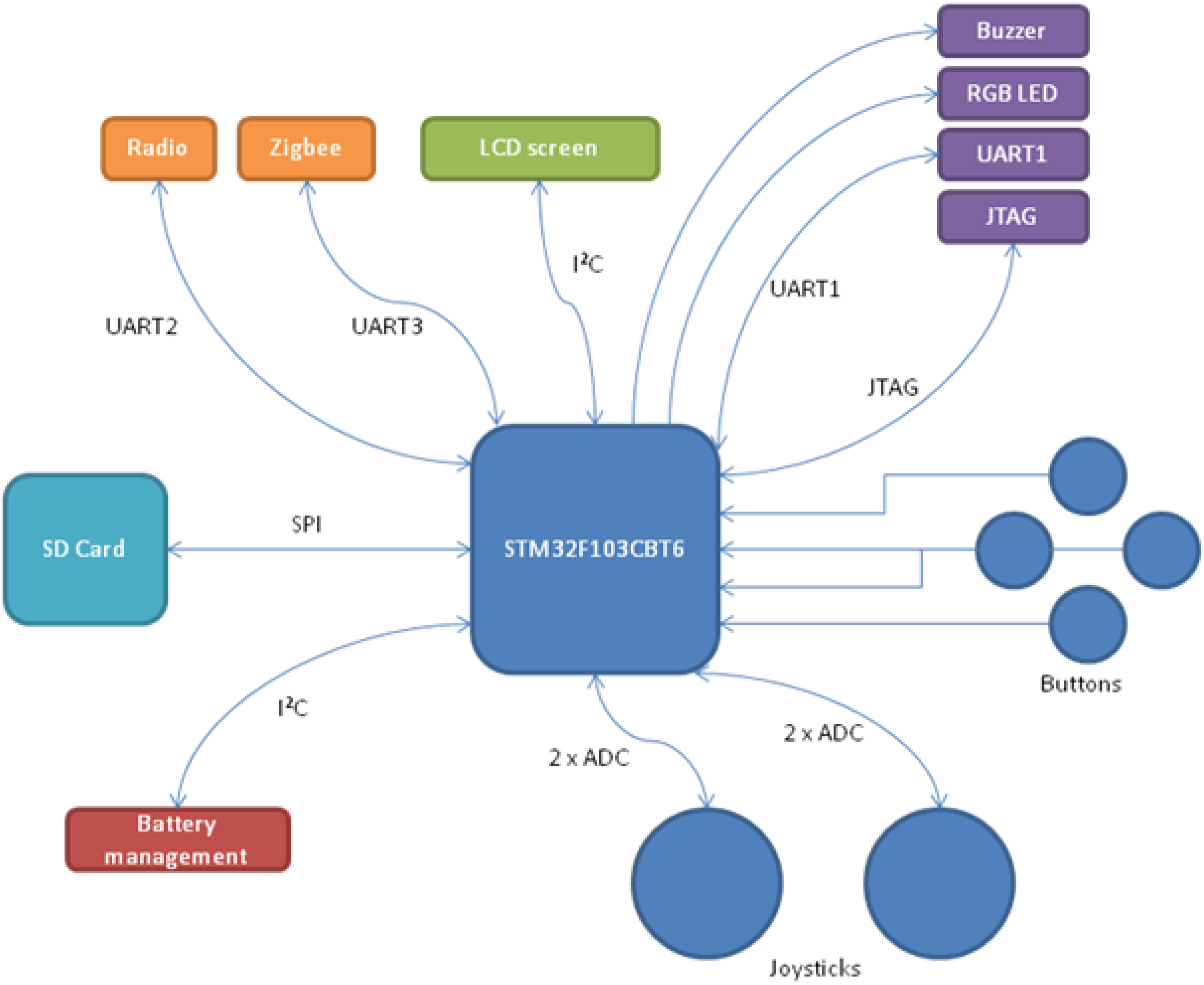


Figure 2.2: Global overview of the board schematics

Chapter 3

Software development

As mentioned before, the software running on the radio interface (PIC16LF876A) was not part of this project. It was provided by Alessandro Crespi and flashed on this microcontroller.

An important part of the project was the implementation of the software on the STM32F103CBT6. It has been first developed on the prototype version of the hardware.

This chapter is dedicated to this software, will explain its functioning and how to modify the code for further development.

3.1 ChibiOS/RT

On the previous version of the remote control, everything was sequential; it did not have the possibility of doing several tasks at a time. Fortunately, it is possible to run on the STM32F103CBT6 a powerful, quite lightweight, easy to use real time operating system named ChibiOS/RT (http://www.chibios.org/, Chibi means little in Japanese).

This offers a huge panel of possibilities in the development of the software and many things can be easily improved. For example, while connecting to a robot, the remote control can also display a scrolling text on the LCD screen. But the main reason for using an OS is that it really eases the development a lot. Indeed, there are some functions already implemented, like the connection to an SD card for example. Moreover, the code of ChibiOS/RT is well typed and helps the developer to make an easily readable and understandable code.

3.1.1 How to use it

First of all, the developer has to install the arm-none-eabi toolchain (version 4.7 (2012-04) used in the project) and then write at the end of its .bashrc the line: export $PATH=$PATH:installation_path. Then, download a version of ChibiOS/RT (version 2.4.2 used in the project).

Define the electrical characteristics of your board using as an example the board defined in the folder boards/ of ChibiOS/RT. Finally, to begin a project, it is highly recommended to start from the examples you can find in the folder testhal/STM32F1xx/ of ChibiOS/RT. Check the configuration of the files mcuconf.f, halconf.h, chconf.h, Makefile and write the code in main.c. These configuration files are very important (definition of the PLL multiplier; what buses are going to be used, at what default speed...).

Finally, to flash the STM32, plug the JTAG probe on the board and the USB on your computer, then define correctly a configuration file openocd.cfg where your Makefile is located. Launch openocd as root user. On another terminal, write:

```
telnet localhost 4444
>halt
>flash write_image  erase build/....elf
>reset init
```

The STM32 is now flashed.

If a developer wants to use the code of this project, the board is defined in the folder boards/UReCon/ of ChibiOS/RT (UReCon stands for Universal Remote CONtrol), the code and the Makefile are in the folder src/. To flash the microcontroller, just launch sudo openocd in the src/ folder and then type make launch in the same folder. During this project, the JTAG interface used was an Olimex ARM-USB-OCD-H which is an *"High speed 3-IN-1 fast USB ARM JTAG, USB-to-RS232 virtual port and power supply 5VDC device (supported by OpenOCD open source arm debugger)"* (https://www.olimex.com/).

3.1.2 What does it provide?

As said above, ChibiOS/RT provides the possibility of doing several tasks at a time, using threads with different priority for example. Moreover, this OS provides clear functions, which enables the developer to write a clean code. For example, sending Hello world ! on the serial port 1 is as this:

```
#include "ch.h"
#include "hal.h"
#include "chprintf.h"
int main(void)
{
        sdStart(&SD1, NULL); //Serial Driver 1, default configuration
        chprintf((BaseChannel *) &SD1, "Hello World !");
        while(1) {
        }
}
```

3.2 Structure of the code and the state machine

3.2.1 Structure of the code

The code of the remote control is structured so that one physical peripheral has its own file with its driver in it. For example, in the file buttons.c can be found the definition of the interrupt handler of each button with a debouncer.

At the beginning, the main function calls the function buttons_init so that the user can control the remote control and then it enters the function menu_init. This function is the one initializing all the working threads and the content of the menus. Among these threads, the one called menuThread contains the state machine of the remote control.

This state machine contains 14 states and almost each state has its own table of structure MENU_ENTRY:

```
typedef struct {
  char menu_buffer[BUFFER_DEPTH];
  MENU next_menu;
} MENU_ENTRY;
```

The table of MENU_ENTRY of a state will thus contain all the choices the user have in the current menu and the state where the state machine gets in after pressing the right button of the remote control on the current entry.

To avoid problems of synchronization, binary semaphores are used so that a pressed button cannot be taken into account if the state machine is processing from one state to another.

Each time the state machine wants to display a new menu entry, it writes the message of the menu in a buffer dedicated to a line of the LCD screen. The thread handling the line display of the LCD screen will display the text and make it scroll horizontally if it does not fit in the 16 displayable characters. The copy of one buffer to another is also protected with a binary semaphore in order to prevent simultaneous accesses on a common resource. The interesting fact is that there are two threads for the LCD screen display handling, one per line. Therefore, the displayed texts can scroll independently. The synchronization of the two threads for the access to the I2C bus is made by using the i2cAcquireBus and i2cReleaseBus functions provided by ChibiOS/RT in the LCD display functions defined in lcd.c.

3.2.2 The state machine

Thirteen states have defined in this quite complex state machine:

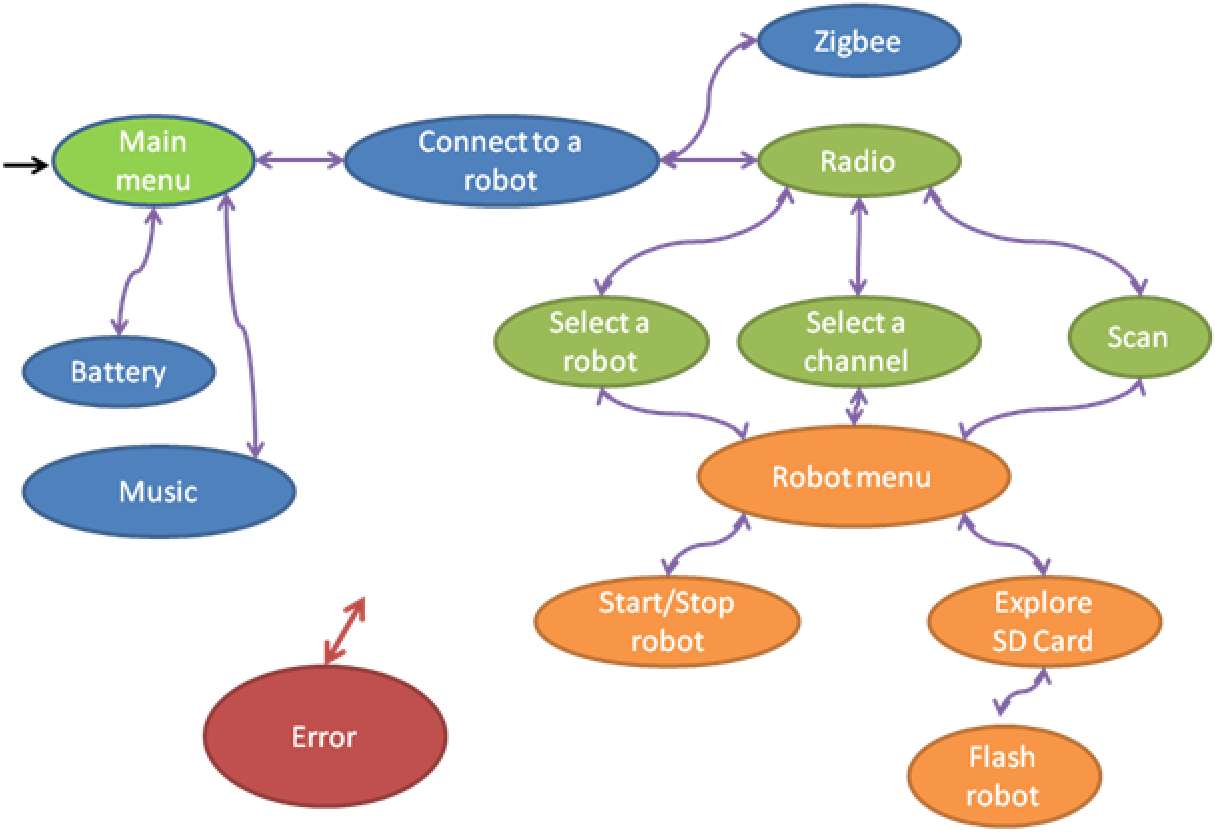


Figure 3.1: State machine used in the remote control

1. MAIN: This is the state the remote control enters when it is turned on. It gives to possibility to the users to either connect to a robot, or display the battery state, or play music with the buzzer. This is a state which can be accessed at any time in every menu when the user chooses the entry *"Back to main menu"*.

This entry will thus not be mentioned in the definition of the other state, to avoid redundancy.

2. BATTERY: This state enables the user to see the information the battery management unit (DS2764) sends to the microcontroller.
3. MUSIC: From this state, the user can play recorded songs on the buzzer, though the sound of it is not fantastic.
4. CONNECTION: If the user selected *"Connect to a robot"* in the main menu, it gets in this one and has the choice between the radio communication (custom protocol over a 868MHz radio link provided by the Nordic nRF905 chip) and the zigbee communication.
5. RADIO: If the radio communication was chosen in the previous state, the state machine goes to this one where the user has several possibilities for connecting to a robot.
6. SELECT_CHANNEL: The user can select manually the channel of the robot he wants to connect to.
7. SCAN: Once entered in this state, the remote control starts scanning and stops when it finds a robot. The user can then decide to select this robot or continue the scan.
8. SELECT_ROBOT: Enables the user to connect to a pre-recorded robot channel. This list can be edited the function menu_init.
9. ROBOT: The state machine goes to this state only if the connection with the robot has been successful. From there, the user can either start the robot of flash it.
10. ROBOT_STARTING: This is just a transition state to start the robot and then get in the next state.
11. ROBOT_STARTED: Once the robot is started, another thread is defined which gets the value from the joysticks using the ADC driver. The thread gets the values of the joysticks' positions and updates the remote robot with these values in order to control it. From this state, the user can only stop the robot.
12. FLASHING: Once this state is entered, the SD Card is connected and mounted using the FatFS library. The user can browse completely the SD Card, list the folders and the files and go from a folder to another. Once a file with an extension .bin is selected, the remote control tries to flash the robot with it.

13. ERROR_STATE: This state can be accessed at any moment if an error is encountered (radio communication error, flashing error...). This state displays the complete error message on the LCD screen.

3.3 Radio communication

The radio interface communicates with the STM32 via the UART2 connection. Establishing communication consists in merely writing and reading registers. These operations are defined in the file radio_reg_op.c. It must be the remote control which initiates the communication as the robot can't, he is a complete slave. When a register operation is performed, the radio interface confirms the operation with either an acknowledgment (ACK) in case of success or with a negative acknowledgment (NACK) in case of failure.

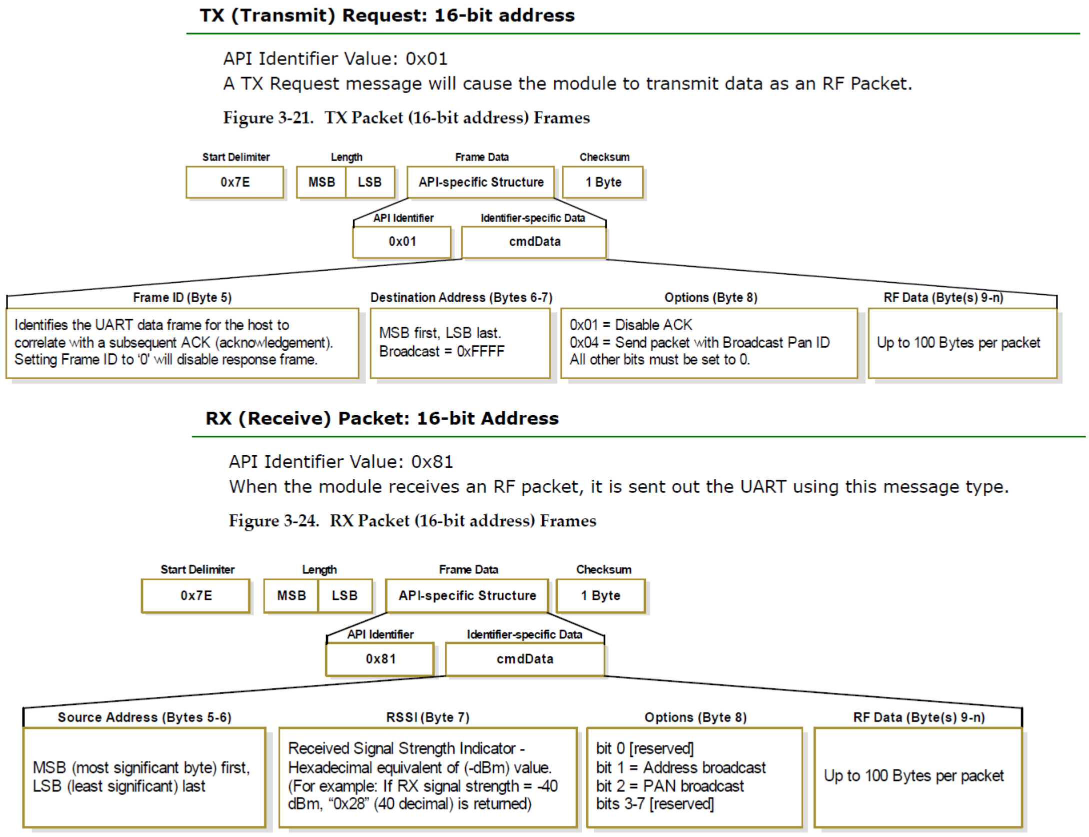
Before using the radio interface, the microcontroller has to synchronize with it, by sending 0xFF for a while, and then sending a character and see if the radio interface answers the same character.

Once in the ROBOT state of the state machine, if the user selects *"Start the robot"* in the menu, a new thread is created, which reads the value of the joysticks and send them to the robot in order to control it. The thread stops when the user leaves the ROBOT_STARTED state.

3.4 Zigbee communication

A zigbee driver was implemented, though not tested, and has to be a little bit adapted according to which kind of module is used. Indeed, the XBee S2 modules use a complex network. There must be one coordinator which communicates with communicates with other XBee modules which are either router or end device. The other kind of module is really simpler, it just a peer to peer network. Similarly to the radio interface, the zigbee module communicates via an UART connection with the microcontroller.

The driver implemented uses the API Frame of the Xbee module which a quite clean way to communicate. Structured packets are sent and received and this enables to have a nice protocol of communication.



TX (Transmit) Request: 16-bit address

API Identifier Value: 0x01
A TX Request message will cause the module to transmit data as an RF Packet.

Figure 3-21. TX Packet (16-bit address) Frames

Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x01
Identifier-specific Data
cmdData
Frame ID (Byte 5)
Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). Setting Frame ID to '0' will disable response frame.
Destination Address (Bytes 6-7)
MSB first, LSB last. Broadcast = 0xFFFF
Options (Byte 8)
0x01 = Disable ACK
0x04 = Send packet with Broadcast Pan ID
All other bits must be set to 0.
RF Data (Byte(s) 9-n)
Up to 100 Bytes per packet

RX (Receive) Packet: 16-bit Address

API Identifier Value: 0x81
When the module receives an RF packet, it is sent out the UART using this message type.

Figure 3-24. RX Packet (16-bit address) Frames

Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x81
Identifier-specific Data
cmdData
Source Address (Bytes 5-6)
MSB (most significant byte) first, LSB (least significant) last
RSSI (Byte 7)
Received Signal Strength Indicator - Hexadecimal equivalent of (-dBm) value. (For example: If RX signal strength = -40 dBm, "0x28" (40 decimal) is returned)
Options (Byte 8)
bit 0 [reserved]
bit 1 = Address broadcast
bit 2 = PAN broadcast
bits 3-7 [reserved]
RF Data (Byte(s) 9-n)
Up to 100 Bytes per packet

Figure 3.2: Excerpt from the Xbee module user manual about the API frame

3.5 Remote flashing

When the user is in the ROBOT menu, it has the possibility to flash the robot the remote control is connected to. If the user chooses to do it, the remote control enters the FLASHING state. In this state, the remote control has first to connect to the SD Card. ChibiOS/RT implements a driver made for the connection of memory cards which is called MMC over SPI driver. This driver is quite useful as it really simplifies the code, all the SPI communication is done automatically. It is a real good example of how ChibiOS/RT is powerful, easy to use and provides an easily understandable code. Here is the code for connecting an SD card:

```
mmcObjectInit(&MMCD1, &SPID2, &ls_spi2cfg, &hs_spi2cfg,
mmc_is_protected, mmc_is_inserted);
mmcStart(&MMCD1, NULL);
chThdSleep(100);

if(mmcConnect(&MMCD1)) {
        send_error_msg("SD Card needed...", error_msg);
        return 0;
}
```

MMCD1 is the memory card driver, SPID2 is the SPI2 driver, ls/hs_spi2cfg stands for low speed/high speed SPI2 configuration and the two last arguments of mmcObjectInit are the function reading the pads connected to the Card Detect and the Write Protect pad of the SD slot.

Once the SD card is connected, the remote control uses the FatFS library (http://elm-chan.org/fsw/ff/00index_e.html) to mount the SD card and then works with files and folders. This library has been chosen for many reasons. First of all, it enables the remote control to read an SD card which is using FAT file system which is quite common. Examples can be found in the ChibiOS/RT folder. Finally, it uses clear functions for dealing with files and folders, quite similar to the usual functions in C (f_open instead of fopen).

After mounting the SD card (f_mount), the remote control displays a kind of Windows Explorer which enables the user to browse completely the SD card. The up and down buttons makes the user go through all the files and folders of a current folder. The left and right buttons enables the user to get in a folder or get in the previous folder. Once the firmware is selected by the user, the flashing can start.

Flashing is quite simple; the remote control enters the bootloader of the robot, opens the firmware file, prepares the flash sectors of the remote robot, erases them, writes parts of the firmware on the RAM and then copies the RAM content to the flash sector, and do it until reaching the end of the firmware file.

Chapter 4

Mechanical integration

One of the main objectives of this project was to provide at the end a case more ergonomic than the one made by G. Cuendet. Whereas G. Cuendet bought a case and adapted it to make the housing of the PCB, the case of this project was firstly designed in 3D using SolidWorks and printed using a 3D printer.

4.1 Inspiration for the new case

As explained in the introduction, the case of the remote control made by G. Cuendet barely fits in one hand and the user has to grab the joystick with two fingers in order to move it. However the position of the four buttons is quite relevant as they are disposed in the same way they are on a game controller for example.

This consideration gave the inspiration of the case of the remote control. Indeed, a game controller for a game station is generally quite ergonomic. If it was not, not so many people would buy it and enjoy playing with it so much. What caught the attention for this case was the game controller of the Sony Playstation. It has been the same for all the three versions; it is to say almost 20 years, which is quite impressive. Taking that figure into account, the reason why Sony did not change it might be that the game controller is very ergonomic. It is quite true as there are two handles to hold the controller which fits perfectly with the hand and then when the controller is hold, the thumbs can easily access to the buttons and the joysticks.

Therefore, a SolidWorks template of the Sony PS3 game controller, found on http://www.3dcontentcentral.com/, was used to design the final case on SolidWorks.

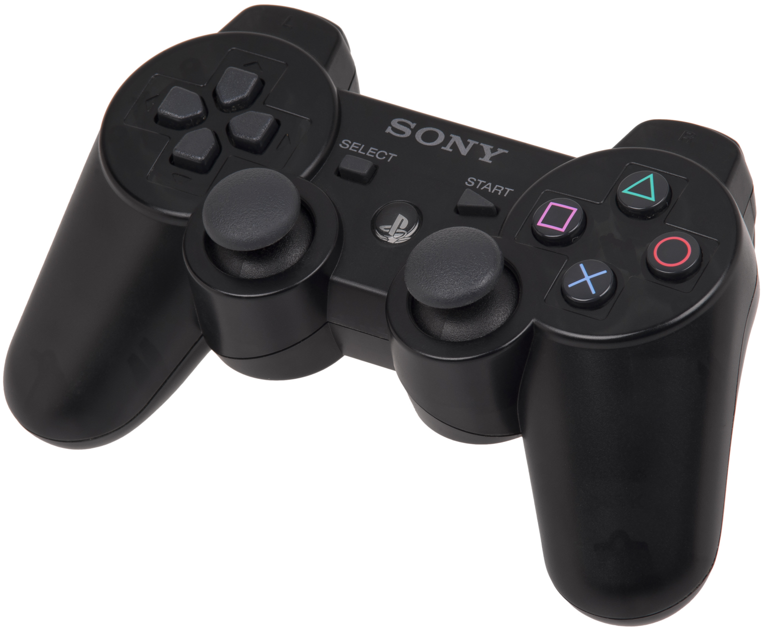


Figure 4.1: Sony Playstation 3 game controller (Source: http://www.videograndpa.com/)

4.2 SolidWorks design

This template was adapted by removing the buttons on the left side and behind and increasing the inner space during the behind buttons so that the final PCB can fit.

Several holes were added so that the LCD screen can be integrated on the top left side and to give access to the different connectors: SD slot, antenna, power jack and on/off switch. A slot for the battery has also been designed on the bottom part of the case.

The 3D model of the PCB generated by Altium Designer allowed to give the right dimensions to the case and to place fixations at the right spots. Once the SolidWorks model was completely designed, an HP 3D printer was used to make the case.

While this report is being written, the final PCB is in production. Thus, only the different parts of the case will be shown in this report.

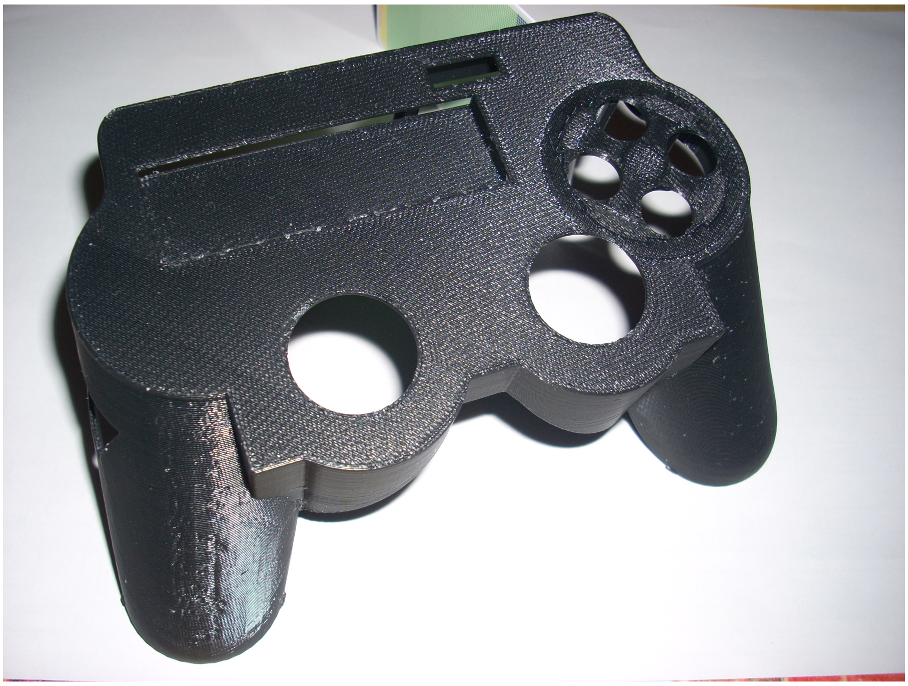


Figure 4.2: Top part of the 3D printed remote control

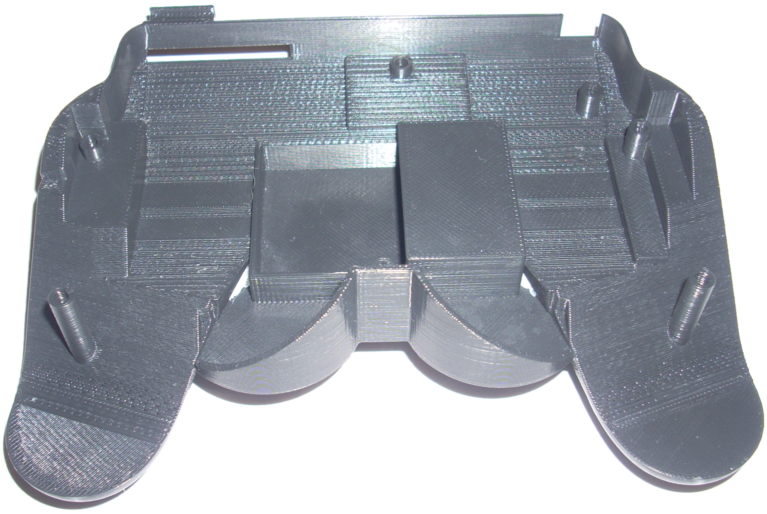


Figure 4.3: Bottom part of the 3D printed remote control

Chapter 5

Conclusion

The remote control made during this project almost meet all the objectives defined at the beginning. It is really more ergonomic as the case was designed and printed. It can flash a robot (that supports this function) with an SD card through the radio communication. However, the designed software only implements the control of the Salamandra robotica. Time was missing to implement the control of the other robots but the code was made so that it is easy to add a new control and it might be implemented even after the end of the project when the new PCB is received in order to provide a complete device for the Biorobotics laboratory.

Nevertheless, this remote control is not perfect and could be improved. For instance, there is a zigbee driver coded but tests should be made in order to control the robot communicating via zigbee with the remote control. Moreover, a universal protocol could be implemented so that the remote control would be able to control any robot in the lab, even the new ones. It could be based on a frame system, as implemented in the zigbee driver. With this protocol, it would be the code of the robot which would have to comply with the data sent by the remote control.

A micromatch for I2C communication has been placed on the final PCB so that an IMU could be plugged. The implementation of a Kalman filter would then enable the remote control to be used in the same way that the Nintendo Wii-Mote. The FreeIMU (Varesano) is recommended for such a further development because it is rather cheap, well documented and functioning code is provided.

Chapter 6

Appendix

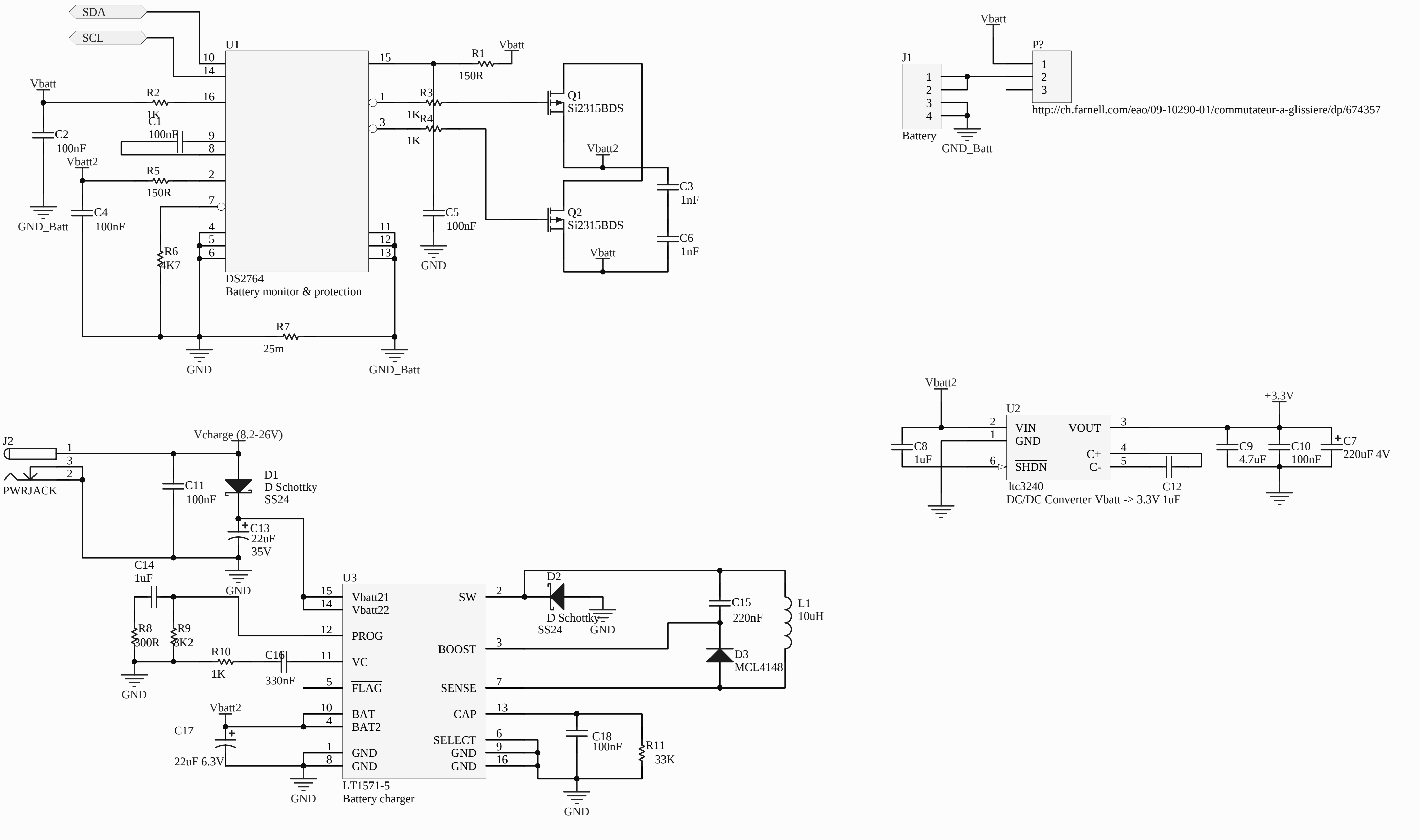
6.1 Pin mapping

6.2 Schematics

6.3 PCB 3D models

6.4 SolidWorks drawings

LQFP48	LQFP64	5V Tol.	PRIM	ALT DEFAULT				ALT REMAP		TYPE	RESULT	
1	1		VBAT							S		
2	2		PC13	TAMPER-RTC						IO	Bouton 1	
3	3		PC14	OSC32_IN						IO	Bouton 2	
4	4		PC15	OSC32_OUT						IO	Bouton 3	
5	5		OSCIN					PD0		I		
6	6		OSCOUT					PD1		O		
7	7		RSTn							IO		
8	12		VSSA							S		
9	13		VDDA							S		
10	14		PA0	WKUP	USART2_CTS	ADC12_IN0	TIM2_CH1_ETR			IO	Joystick 1 Axe X	
11	15		PA1		USART2_RTS	ADC12_IN1	TIM2_CH2			IO	Joystick 1 Axe Y	
12	16		PA2		USART2_TX	ADC12_IN2	TIM2_CH3			IO	RF	
13	17		PA3		USART2_RX	ADC12_IN3	TIM2_CH4			IO	RF	
14	20		PA4	SPI1_NSS	USART2_CK	ADC12_IN4				IO	Reset RF	
15	21		PA5	SPI1_SCK		ADC12_IN5				IO	Joystick 2 Axe X	
16	22		PA6	SPI1_MISO		ADC12_IN6	TIM3_CH1	TIM1_BKIN		IO	Joystick 2 Axe Y	
17	23		PA7	SPI1_MOSI		ADC12_IN7	TIM3_CH2	TIM1_CH1N		IO	LED	
18	26		PB0			ADC12_IN8	TIM3_CH3	TIM1_CH2N		IO	LED	
19	27		PB1			ADC12_IN9	TIM3_CH4	TIM1_CH3N		IO	LED	
20	28	oui	PB2/BOOT1							IO		
21	29	oui	PB10	I2C2_SCL	USART3_TX			TIM2_CH3		IO	Zigbee	
22	30	oui	PB11	I2C2_SDA	USART3_RX			TIM2_CH4		IO	Zigbee	
23	31		VSS_1							S		
24	32		VDD_1							S		
25	33	oui	PB12	SPI2_NSS	USART3_CK	TIM1_BKIN	I2C2_SMBALERT			IO	SD Card	
26	34	oui	PB13	SPI2_SCK	USART3_CTS	TIM1_CH1N				IO	SD Card	
27	35	oui	PB14	SPI2_MISO	USART3_RTS	TIM1_CH2N				IO	SD Card	
28	36	oui	PB15	SPI2_MOSI		TIM1_CH3N				IO	SD Card	
29	41	oui	PA8		USART1_CK	TIM1_CH1	MCO			IO	Buzzer	
30	42	oui	PA9		USART1_TX	TIM1_CH2				IO	UART	
31	43	oui	PA10		USART1_RX	TIM1_CH3				IO	UART	
32	44	oui	PA11	CANRX	USART1_CTS	TIM1_CH4	USBDM			IO	SD Card CD	
33	45	oui	PA12	CANTX	USART1_RTS	TIM1_ETR	USBDP			IO	SD Card WP	
34	46	oui	JTMS/SWDIO	PA13						IO	JTAG	
35	47		VSS_2							S		
36	48		VDD_2							S		
37	49	oui	JTCK/SWCLK	PA14						IO	JTAG	
38	50	oui	JTDI	PA15				TIM2_CH1_ETR	SPI1_NSS	IO	JTAG	
39	55	oui	JTDO	PB3	TRACESWO			TIM2_CH2	SPI1_SCK	IO	JTAG	
40	56	oui	JNTRST	PB4				TIM3_CH1	SPI1_MISO	IO	JTAG	
41	57		PB5			I2C1_SMBALERT		TIM3_CH2	SPI1_MOSI	IO	Bouton 4	
42	58	oui	PB6	I2C1_SCL		TIM4_CH1		USART1_TX		IO	Battery Monitoring	IMU/LCD
43	59	oui	PB7	I2C1_SDA		TIM4_CH2		USART1_RX		IO	Battery Monitoring	IMU/LCD
44	60		BOOT0							I		
45	61	oui	PB8			TIM4_CH3		I2C1_SCL	CANRX	IO	Bouton Joystick 1	
46	62	oui	PB9			TIM4_CH4		I2C1_SDA	CANTX	IO	Bouton Joystick 2	
47	63		VSS_3							S		
48	64		VDD_3							S		



Title *Alimentation*

Size: A4 Number:1 Revision:1

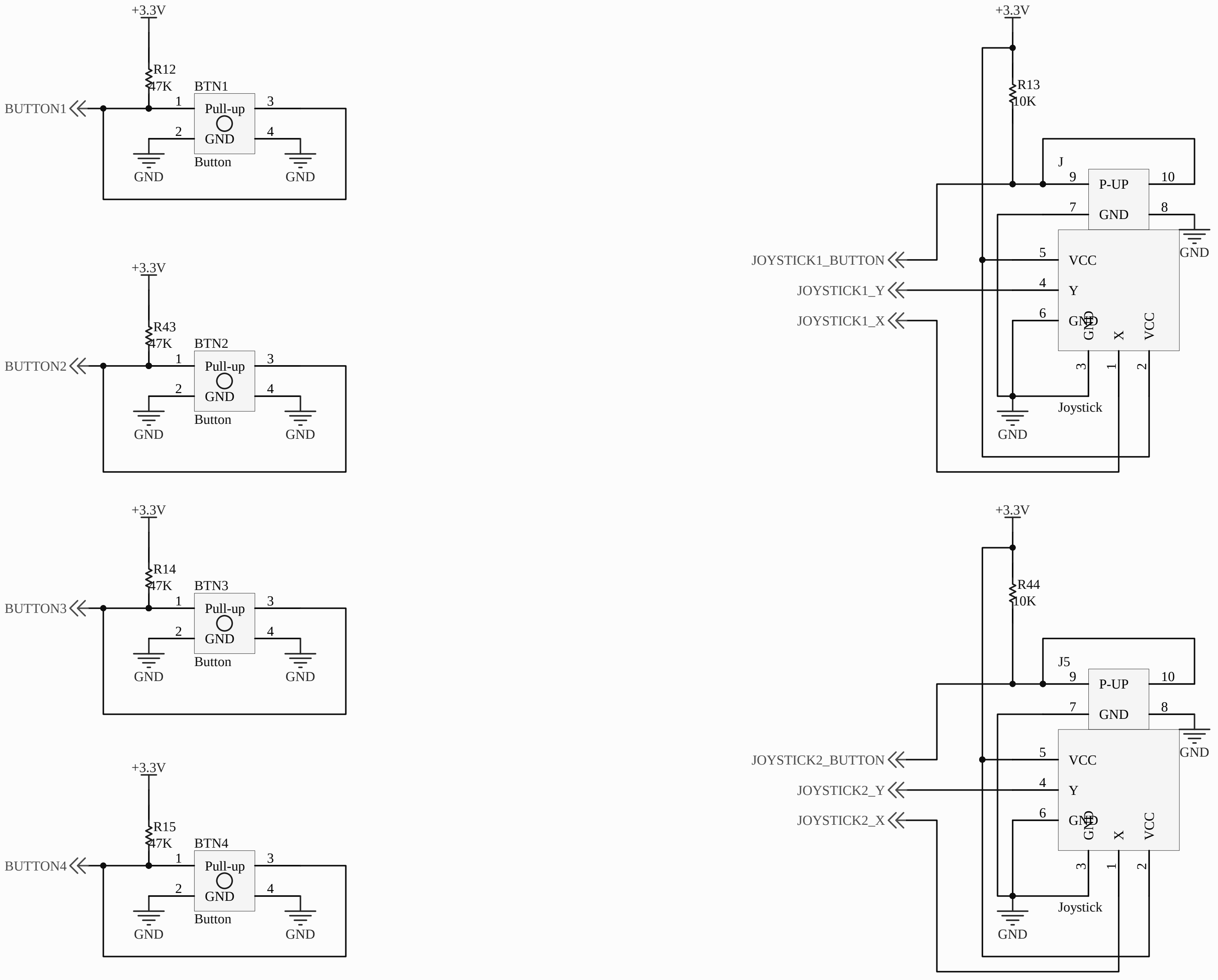
Date: 14/01/2013 Time: 02:06:04 Sheet1 of 5

File: E:\Matthieu\boulot\EPFL\Projet Microtechnique I\PCB_final\alimentation.SchDoc

*

EPFL

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



Title *Buttons*

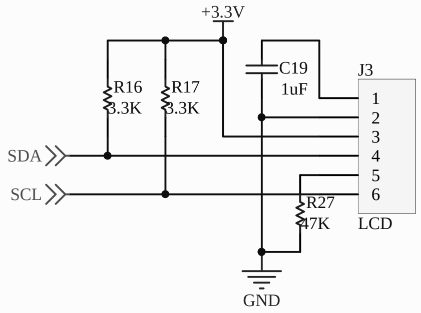
Size: A4 Number:1 Revision:1

Date: 14/01/2013 Time: 02:07:41 Sheet5 of 5

File: E:\Matthieu\boulot\EPFL\Projet Microtechnique I\PCB_final\buttons.SchDoc

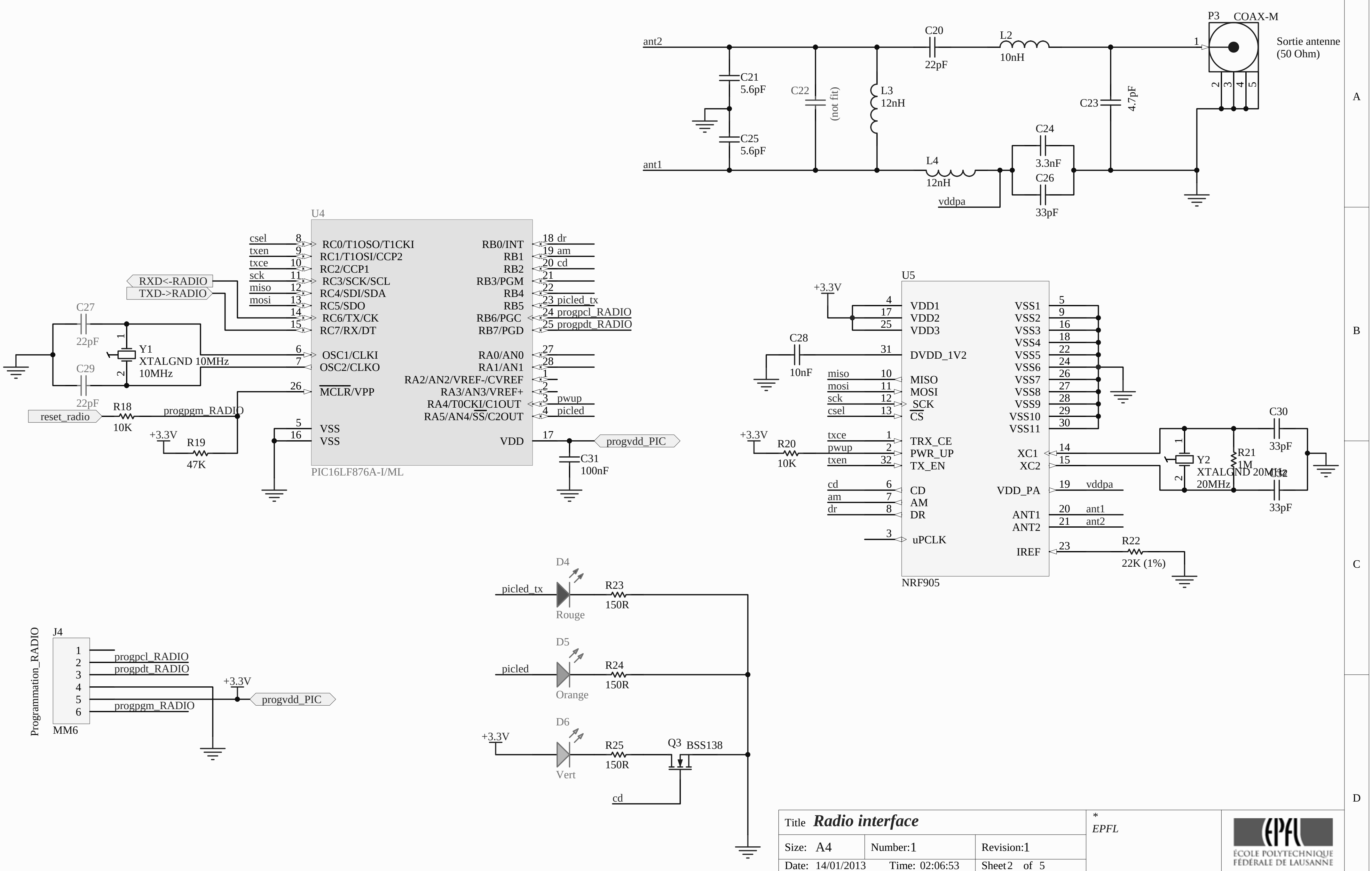
*
EPFL

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



Title ***LCD***			* *EPFL*
Size: A4	Number:1	Revision:1	
Date: 14/01/2013	Time: 02:07:30	Sheet4 of 5	
File: E:\Matthieu\boulot\EPFL\Projet Microtechnique I\PCB_final\lcd.SchDoc			

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE



Title *Radio interface*

*

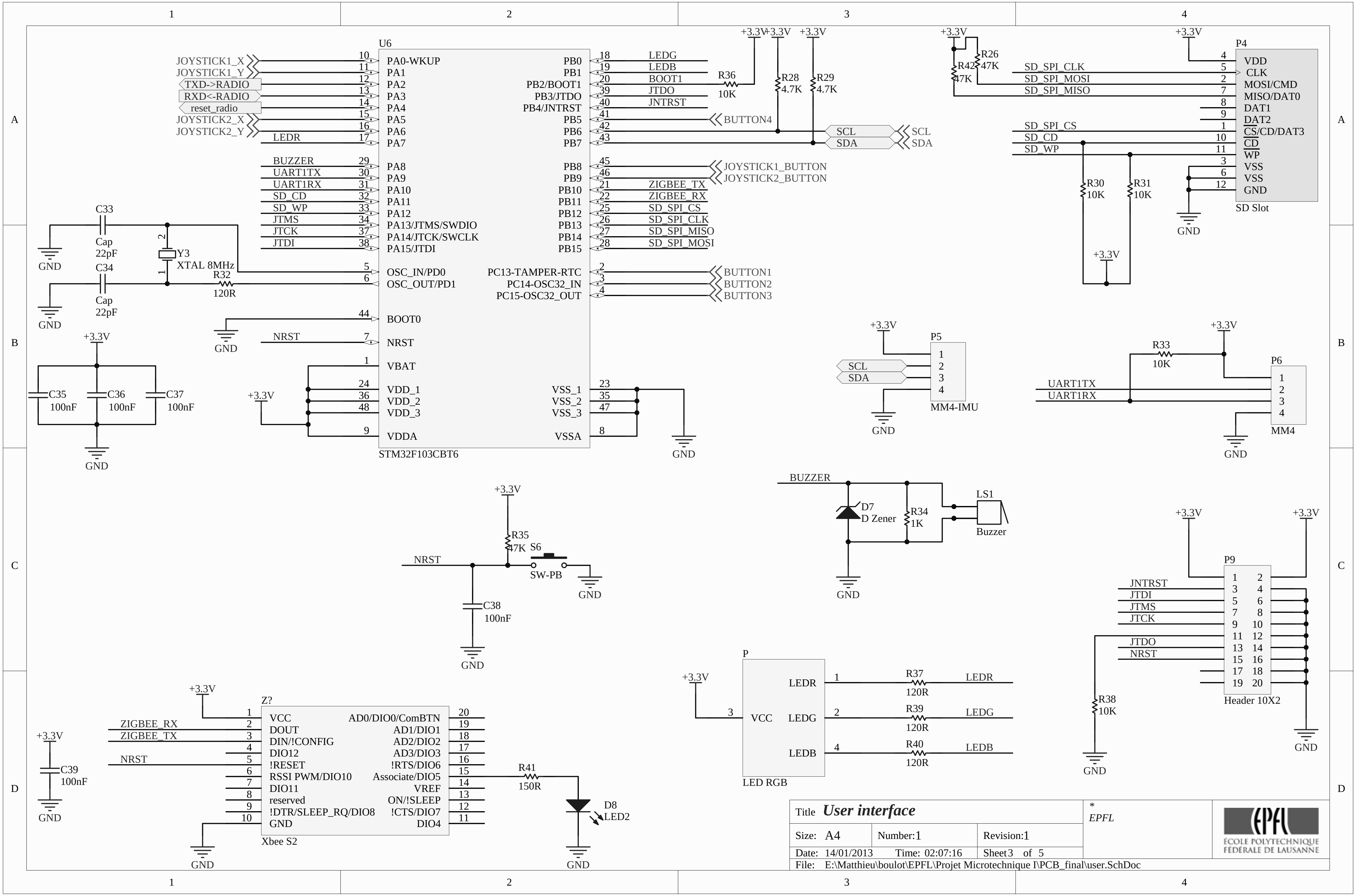
EPFL

Size: A4 Number:1 Revision:1

Date: 14/01/2013 Time: 02:06:53 Sheet2 of 5

File: E:\Matthieu\boulot\EPFL\Projet Microtechnique I\PCB_final\radio.SchDoc

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE



Title *User interface*

Size: A4 Number: 1 Revision: 1

Date: 14/01/2013 Time: 02:07:16 Sheet3 of 5

File: E:\Matthieu\boulot\EPFL\Projet Microtechnique I\PCB_final\user.SchDoc

*
EPFL

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

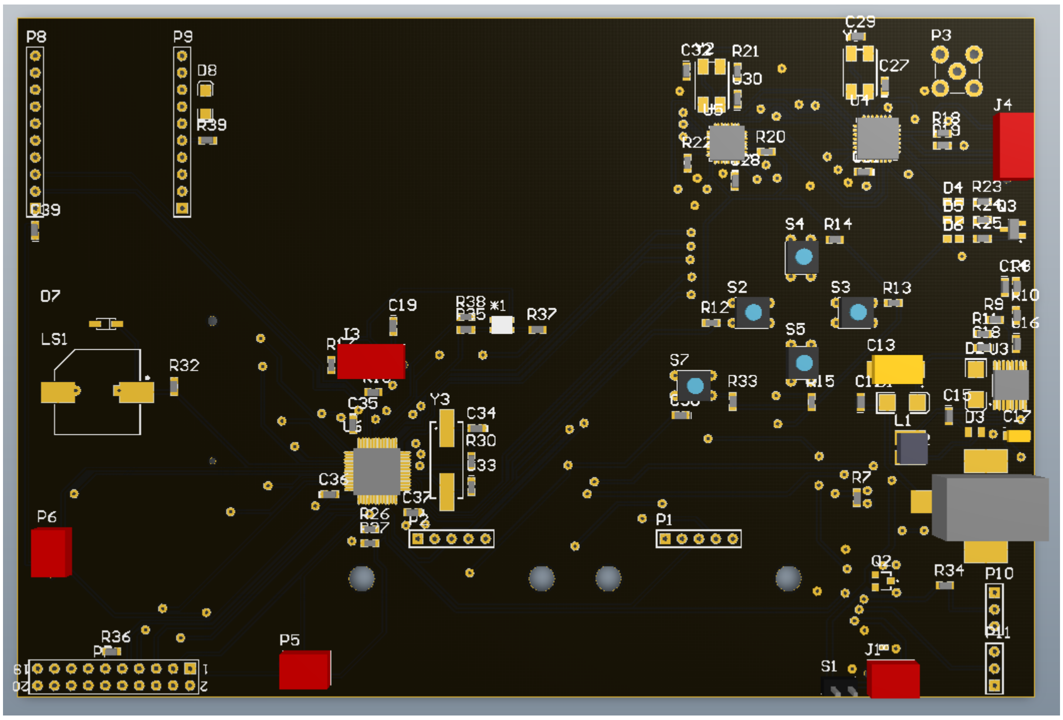


Figure 6.1: 3D model of the PCB prototype

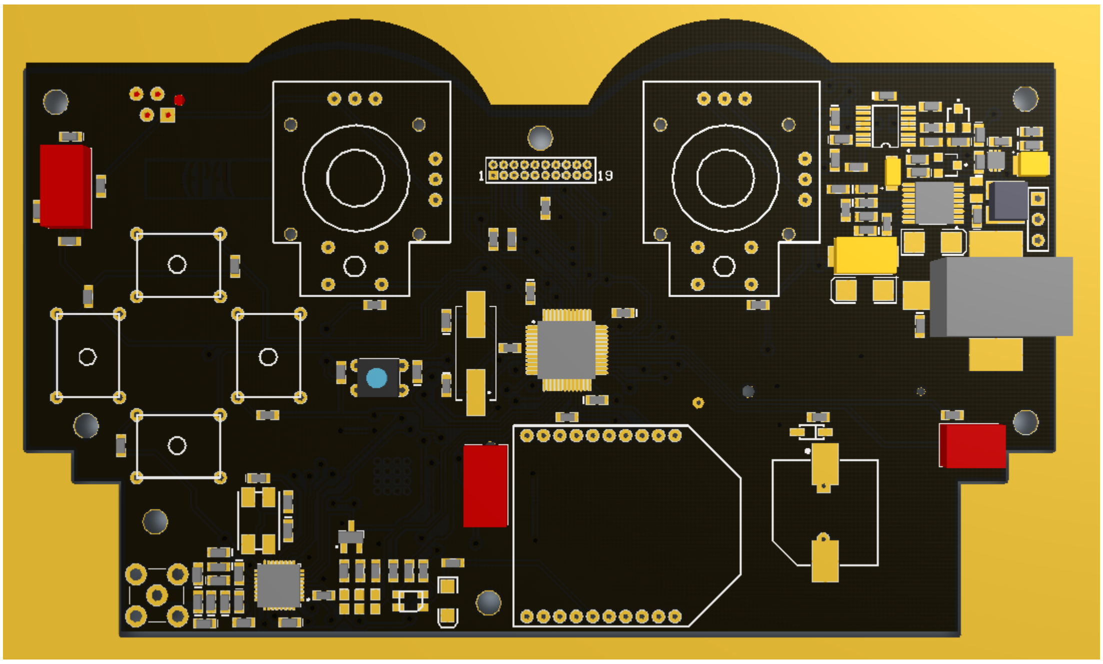


Figure 6.2: 3D model of the final PCB

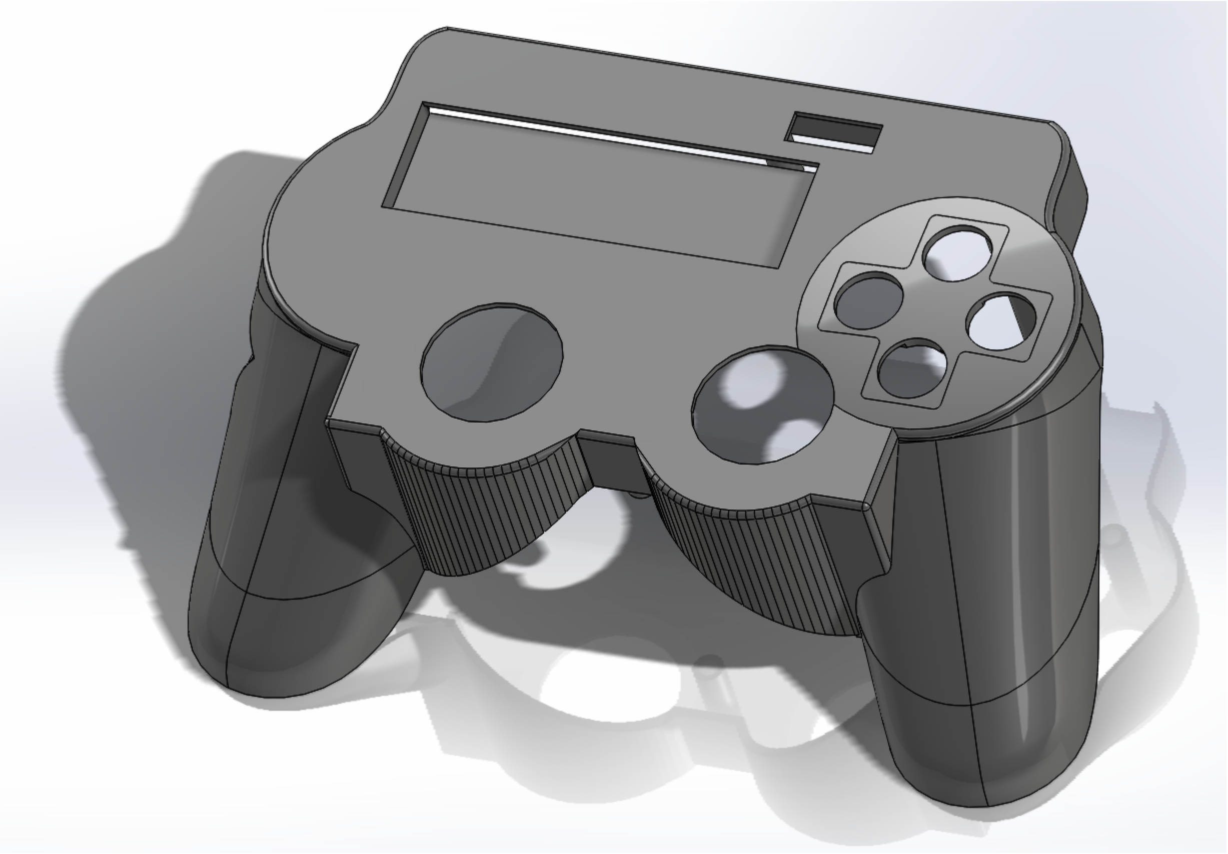


Figure 6.3: Top view of theSolidWorks 3D model of the top part of the remote control

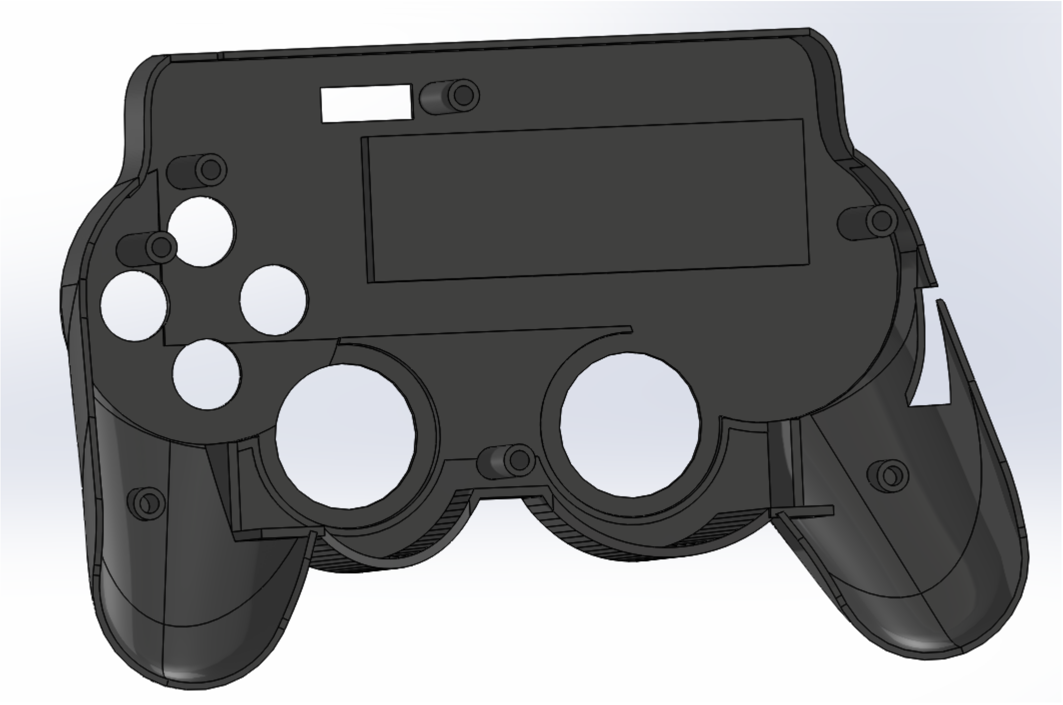


Figure 6.4: Bottom view of theSolidWorks 3D model of the top part of the remote control

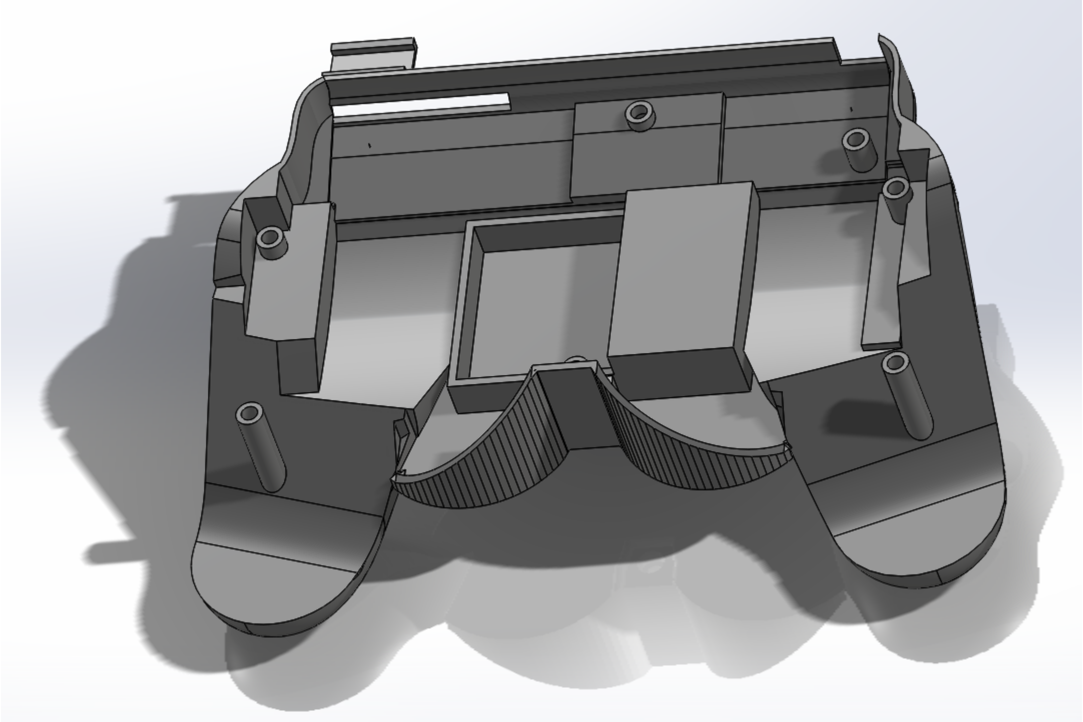


Figure 6.5: Top view of the SolidWorks 3D model of the bottom part of the remote control

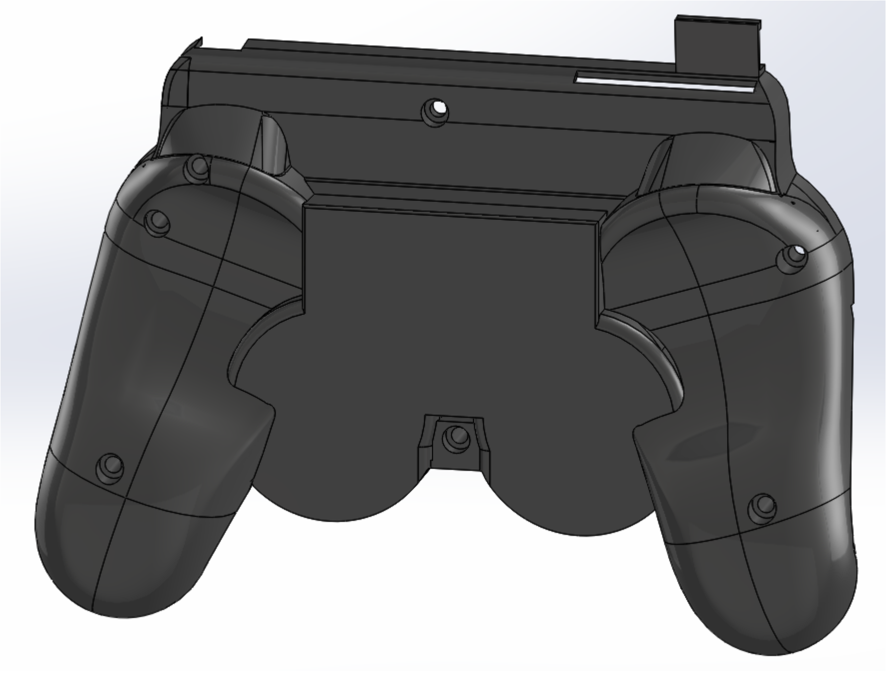


Figure 6.6: Bottom view of the SolidWorks 3D model of the bottom part of the remote control