PLP Project

Create a smart HMI

Antoine Mallet

Engineer Student (i4-SE)

Project in Brunel University

Supervisor: Dr Ali Mousavi

May 2nd to July 28th 2012

Brunel UNIVERSITY WEST LONDON

ESIEE PARIS

Acknowledgements

First and foremost, I'd like to thank my supervisor, Dr Ali Mousavi who helped me to start this research with an open mind and afforded me the broadest possible freedom to choose my research topic and to organize myself.I appreciated his support to understand the bases of human motion modelling. I took full benefits of the discussions we had during the course of this project.

I'd like also to thank Mrs Tatiana Kalganova who helped me to solve the different administrative procedures I had to face before and on my arrival. Thanks to her, this was not an issue to find good accommodation on my arrival, and even for a guest researcher, this is an important starting point.

I'd like also to thank all University colleagues who accompanied me during this 3 months research

Introduction

This project aims to reduce the phantom pain experienced by amputees. Indeed, even long after an amputation, patients may experience severe pain or itching in the member they have lost. The source of this phenomenon remains controversial but it turns out that if the subject is able to see the image of the member being lost (eg by reflecting in a mirror the healthy limb), then the pain subsides.

The above photos illustrate the experience of the mirror box. The goal of this project is to achieve a similar experiment but with digital images and to create scenarios where the movement of the missing limb is independent from the movement of the healthy limb. For example, it would then be possible to use:

- 3D images to improve the rendering
- Sensors (accelerometers, pressure sensors, etc.) located on the stump to determine the movement that the patient wants to achieve with his amputated limb

The ultimate goal is to minimize the phantom pain using as little material as possible to minimize the cost of the equipment.

This project is in the research stage and the proposed topics for my internship were numerous. I chose to work on the modelling of movements of the missing member according to different parameters that I had to determine. My work was divided into several parts:

- Acquisition of the sensor (an accelerometer placed on the stump)
- Data transmission to the computer
- Data analysis and display of the result as a 2D rendering (image of a hand)

The challenges were to:

- Determine a model which I could use with the data I had
- Make the system adaptable to different users and for a specific user
- Minimize the latency to make the system as close as possible to reality

In this report I will name the patient who may use my work as "the user" and the whole electronics material as "the system".

Table of Contents

A - An appropriate model

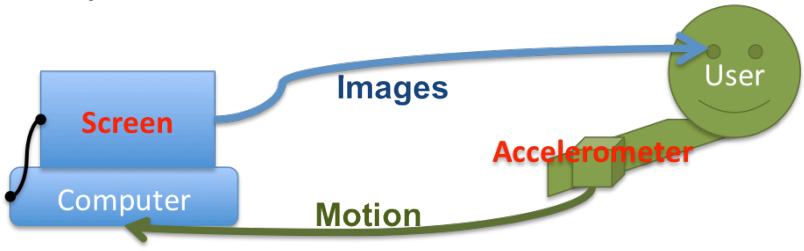
A.1 - Processing data to create a virtual limb

Context

This project included a part of research. Indeed, today the movement of the human body (and in general the movements of living beings) is not an exact science. Obviously, this area is evolving rapidly: remember the bearing of the dinosaurs in the movie *Jurassic Park*, how unnatural they looked compared to today's movie made entirely with computer graphics. In spite of present progress, the formulas describing movement remain empirical and it was necessary to make choices about the algorithms used in the PLP project.
Usually, we determine the position of the limbs according to the movements that have to be performed. For example, in a movie, if a character walks we have to move the legs according to the speed he is supposed to move. This project has two particularities:

- Except if the user is asked to do a specific movement, the system doesn't know the kind of movement the user will do, .
- It's impossible to record the movements of the user's limb so I had to determine them from the movements of an other part of the body. To do that, I thought of three different approaches:
  - Record the motion of the stump or of the still healthy part . *e.g.: Record the motion of the arm to determine to motion of the forearm. Record the motion of the thigh to determine the motion of the leg*.
  - Record the symmetrical limb motion and ask to patient to act as if he moved his arms/legs the same way. Thus it's possible to determine the motion of the phantom limb by doing symmetry of the movement of the healthy limb. This is a kind of a digital reproduction of the experience of the mirror box.
  - Track the user's eyes. *e.g.: To catch an object. In general, when one wishes to catch an object the eye moves towards the object.*

The eye tracking requires a camera and image processing, but the moves of the body can be recorded with an accelerometer or a gyroscope. For this project I decided to use only an accelerometer and get the maximum of workable data from what I can measure. The following diagram shows the interactions between the user and the system.



In a normal situation, one can use **all sensory information** to interact with the surrounding environment and correct or modify the motion path. But in this

simulation, the user can only use his eyesight and the computer has only the accelerometer data. Therefore I distinguish two types of motion: motion with user's feedback control and motion without user's feedback. Indeed, It takes several tenths of second before visual information can affect the execution of a movement. For fast movements, it is inconceivable that the command is modified or adjusted continuously by visual feedback during motor performance. This is particularly true of finger movements for the trained typist whose execution time is around 60 ms.
As an illustration of this statement, the following table gives the processing time of visual information for different types of tasks

Task	Processing time of visual	Work
Alternate movements	500ms	[Woodworth 1999]
Target tracking movements	300~350ms	
Discrete movements of target attainment	190~260ms	[Keele *et al*. 1968]

The consequence is that the user can only perform feedback on slow enough movements . If he makes a quick movement, he will not control the movement continuously, but still will expect to see his hand in the right place on the screen. This kind of movement may be difficult to predict and model correctly however it can give useful information about the user. For example, if the system requests a fast movement of target attainment, then we know the maximum acceleration of the user.

Empirical knowledge

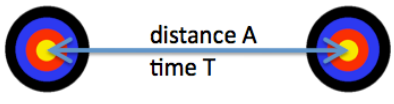
Among the many theories on the decomposition of human body movements, some seemed to be particularly useful for this project.

The earliest studies conducted by Woodworth (1899) and by Fitts (1954), were used to analyze gestures of quick displacement between two targets in the same plane. The experiments conducted by Fitts consisted in executing a back and forth movement between two targets for 20 seconds. The distance between the target and the target sizes changing from one experiment to another. Subjects were also asked to ensure that the precision error does not exceed 5% of the width of the target.
The law expresses the relationship between:

- The duration of motion T
- And the distance A and the precision of motion W

[1]

$$T = a + bI_d = a + b.\log_2\left(\frac{2.A}{W}\right)$$



- Where a and b are constants determined empirically;
- Id characterizes the index of difficulty of the task, expressed in bits.

This equation indicates that the duration of motion increases with the index of difficulty.

Fitts' law is not disputed and verified in very diversified contexts . It actually applies to both children and adults, for movements executed with different effectors. It addressed however, a certain class of movements, which required that the hand reach the extremity point with a great precision.

The expression proposed by MacKenzie (1992) can avoid negative indexes of difficulty, especially for relatively large and nearby targets (i.e. W> 2A). It better reflects the correlation between A and W for certain conditions of experiments.
[2]

$$T = a + b.\log_2\left(\frac{A}{W} + c\right)$$ c=1/2 [Welford 1968] or c=1 [MacKenzie 1992]

Schmidt *et al*. have completed Fitts' law. They are interested in quick movements that prohibit the use of corrections based on sensor feedback. These movements require a reference speed is imposed on the subject. For this type of experiment, there was a linear relationship between speed and precision.
[3]

$$W_e = a + b\frac{A}{T}$$

In this case, the target is reduced to a point and W_e express the deviation between the target and the position of the extremity of the hand.

The principle of isochronie, has been established by Freeman (1914) and reflects the invariance of the duration of execution of the movement as a function of the amplitude of this movement. As shown in Fitts' law, there is indeed a spontaneous tendency to increase speed of movement depending on the distance to travel when no constraints are imposed on the average speed of the gesture.
[4]

$$V = kA^{\alpha}$$

With k and α constant coefficients (α close to 1)

Finally, Meyer *et al*. have shown that Fitts' law can be considered as being a special case of a more general law, known as the law of Meyer (1990).
[5]

$$T = a + b.n\left(\frac{A}{W}\right)^{1/n}$$

where n is the number of sub-movements made to reach a target of size W, at distance A from the initial position of the hand. a and b are constants.

Depending on which model is applied and the scenario chosen by the GUI, one of these equations can be used.

A.2 - Make the system adaptable

I have considered two approaches to make the system adaptable.

- Firstly, it must be able to adapt very specifically to a given user. To do that, we mostly use the above equations in which we try to determine the

value of the many variables. These particular values will be considered as specific to a given user.

- Secondly, it must adapt whatever the user and the environment in which it is used.

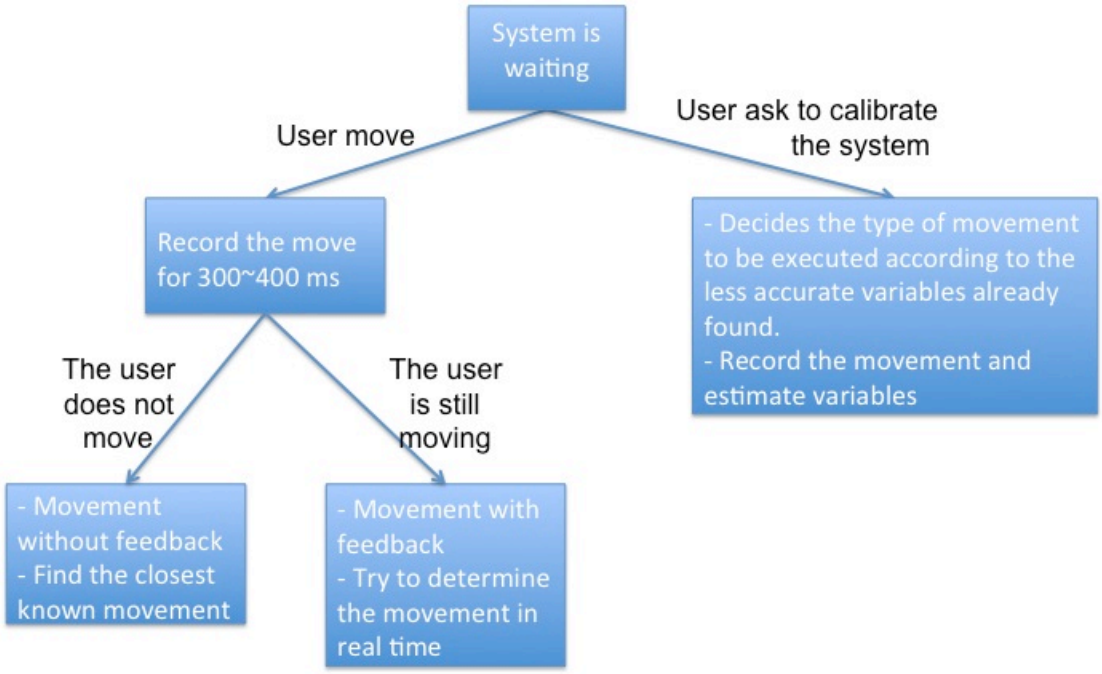
For a specific user

First of all, it's possible to sort the equations according to the two types of motion.

Use of visual feedback	No use of visual feedback (For movement < 500 ms)
[1], [2], [4], [5]	[3], [4]

The two different scenarios are:

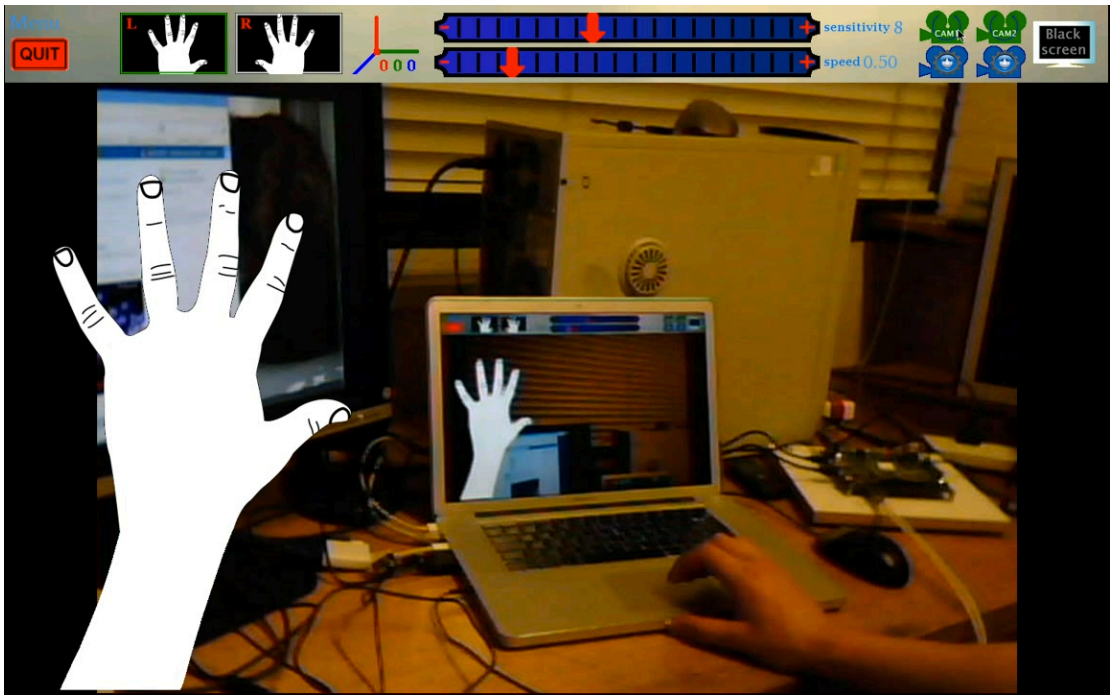
- The user is asked to perform a specified move in order to calibrate the system
- The user is actually using the system, which must then determine his movements.



To different users and environments

The system must be able to adapt whatever the reference user. Obviously the first criterion to consider is the pathology. Therefore, it seemed useful to let the user choose the limb to model. In the graphic interface I did, it is only possible to choose between the right or the left hand, but of course in a complete version it should be possible to choose more accurately (e.g. only the hand, the forearm, the foot).
Than I thought the environment should be integrated. It is therefore possible to take a picture from the HMI, which is then displayed in the background. In an improved version, we can imagine that there would be an image processing to

analyze the various objects present in the picture to give the user the ability to grasp objects he sees.

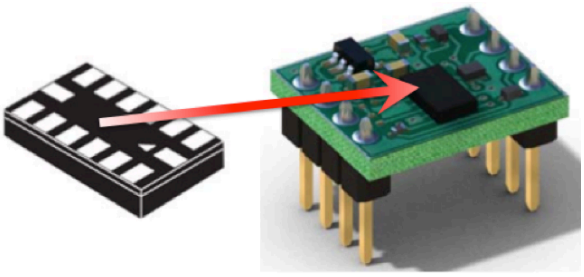


B - The hardware

The aim of this project was to determine a model, the useful parameters to implement this model and the way to do it. I had no specific hardware to use.

The model I selected used time, speed and position, so I had to measure position and speed at a high frequency because some human movement can be quick (~50ms).
At this speed, a camera (30 frame per second) gives only two pictures, probably blurred and need image processing at very high speed, so I kept the accelerometer used by the last year student.

Relevant information about the accelerometer	
Name	MMA7455L
Axes	3 axes: X, Y, Z
Communication protocol	I2C or SPI
SPI frequency	8MHz
Selectable range	±2g ; ±4g ; ±8g
Voltage	2.4V – 3.6V



MMA7455L Parallax module

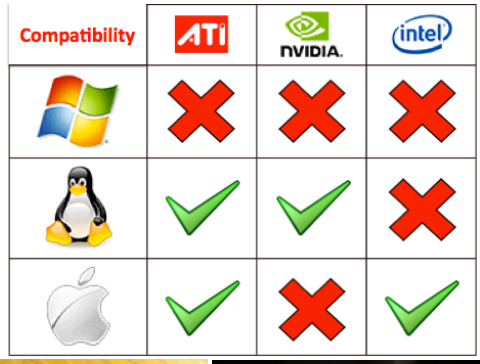
This accelerometer is a MEMS produced by Freescale and sold by Parallax welded on a PCB with a voltage driver which allows it to be used between 2.5V and 5.5V and

some connectors. With this module, the SPI can only be used in a 3-wires mode.

B.1 - Minimum hardware: I2C on VGA

To create a plug-and-play system, which is as simple as possible, I thought about using the ***25¢ I2C Adapter Project***. This project is based on the repurposing of the existing display control lines present on many graphics cards to interface I2C devices directly with a computer without any such adapters as USB-I2C modules that usually cost between 30 and 100$. The controls lines are called DDC (Display Data Channel) and are in fact, simply an implementation of an I^2C bus with a few established rules. By tapping into this connection between the computer and monitor or making use of an unused video port, one can interface with many I^2C devices at virtually no expense and the whole system became as simple as a computer and an I2C device directly connected together.

Unfortunately, this project is a kind of trick with the DDC and has some limitations. First of all, the compatibility range of the library is limited to OSX and Linux and requires specific graphic card. However my laptop had the required configuration so I tried this library on Linux Ubuntu 9.0.4.



I built a simple connector to plug the accelerometer to my VGA port. The proper wiring is

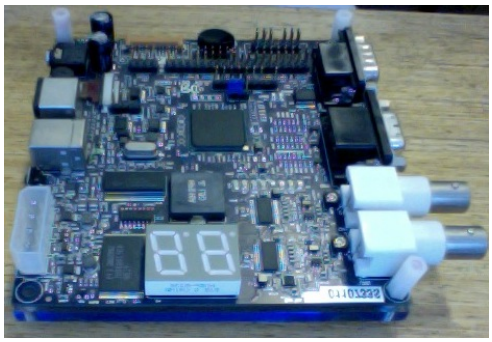
VGA wire	I2C wire
PIN 5 : Ground	Ground
PIN 9 : +5V	VIN
PIN 12 : Data	SDA
PIN 15 : Clock	SCL

However, I was unable to read any data from the accelerometer using this hack. According to the project website, not all I2C device are compatible and I assumed that the MMA7455 wasn't. Moreover, this trick is likely to be a warranty-voiding operation and it would not have given a workable solution. Consequently, I abandoned this idea.

B.2 - Only FPGA: Handel-C

Because the I2C on VGA wasn't a success, I contacted the two students who worked on this project last year in order to get some information on how they acquired data from the accelerometer.
They had made the choice to only use FPGA (2 * Celoxica RC10 board). Thus the data were processed in a very short time and no computer was needed to realise the system. The FPGA was a Xilinx Spartan 3L on the Celoxica RC10 board. This board can be programmed with Handle-C, which simplifies the work and adds distance between the hardware and the code. Unfortunately, nobody was able to give me the work they realised and I would have had to do everything from the beginning if I decided to use this method.
Moreover Celoxica doesn't support the RC10 anymore because Mentor Graphics bought this part of their business and in spite of discussions I had with them, they were not able to provide me with the software needed to use the RC10.

As I could not find the software by myself, I considered that the RC10 was not a good choice anymore and I turned toward the Atlys produced by Diligent, used a Xilinx Spartan 6 FPGA which is optimized for high-performance with fast clock and GPIO.
But in order to make the system adaptable to different users and auto-adaptable to a defined user, I had to create an elaborate graphic interface. It was much easier and user friendly to develop a software with a computer than to add the support of keyboard and/or mouse to the FPGA, so I decided to use only one FPGA which acquires and processes data from the accelerometer before sending them to the computer.

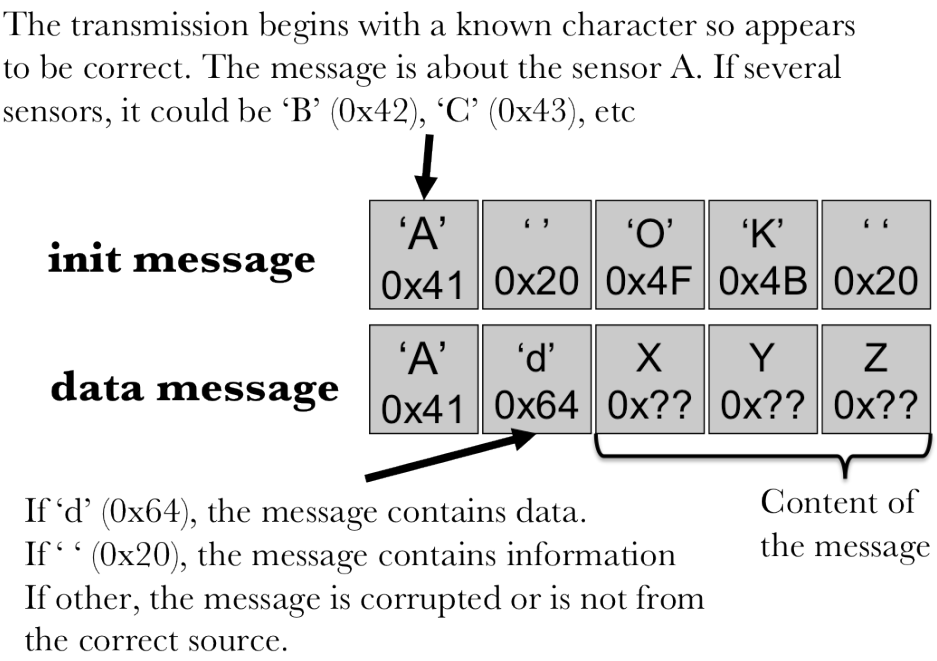


B.3 - FPGA and Interface on Windows

I chose to program the FPGA with Xilinx ISE 14.1 and the graphic interface with the free and cross platform IDE (*Integrated Development Environment*) Code::Blocks on windows.
Some technical details and the instructions to download my work can be found in the "Technical Manual" section.

Acquire data and sending them to the computer

The MMA7455 can communicate faster with SPI than I2C and this protocol is simpler because you only have to place the SC signal to low state to activate the device. So I used SPI and programmed the acquisition from the accelerometer with the FPGA.
To send data to the computer I used the chip XR21V1410 that is embedded on the Atlys board. It is an enhanced 1-channel Universal Asynchronous Receiver and Transmitter (UART) with a USB interface. This means that I could communicate with the serial protocol by plugging in the Atlys with USB. However, this chip only allows the computer to recognize the USB as a serial port and it is still necessary to program the FPGA to write a COM port as any other. I choose to use a baud rate of 9600 because it is a common value that I knew worked with windows and that is sufficient to transmit accelerometer data. Indeed with 9600 baud, 8 data bits (LSB first), 1 stop bit and no parity the duration to transmit a data is one millisecond. As I send the three axes (X, Y, Z) and a first character to indicate that data is valid (or from where it comes) and a second to indicate that this is a data transmission, there is a total of 5 data of 8 bits to transmit. This means that the graphic interface can use new values every 5ms.

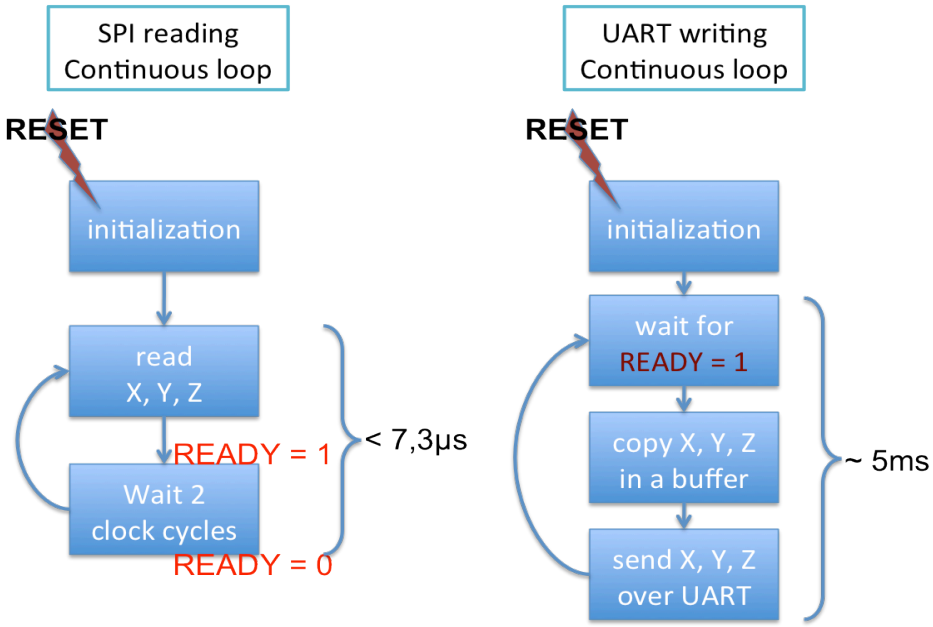


This approach helps making the system adaptable.

- For example if I want to add a second accelerometer or a gyroscope, messages about this new sensor will begin with 'B'.
- Or if I want to add a message informing them that no data is sent from the sensor, this message will begin with '*letter of the sensor*' (0x4?) followed by ' ' (0x20) the content of the message which will be three characters saying that no data is received e.g. "NOd" (0x4E 0x4F 0x64).

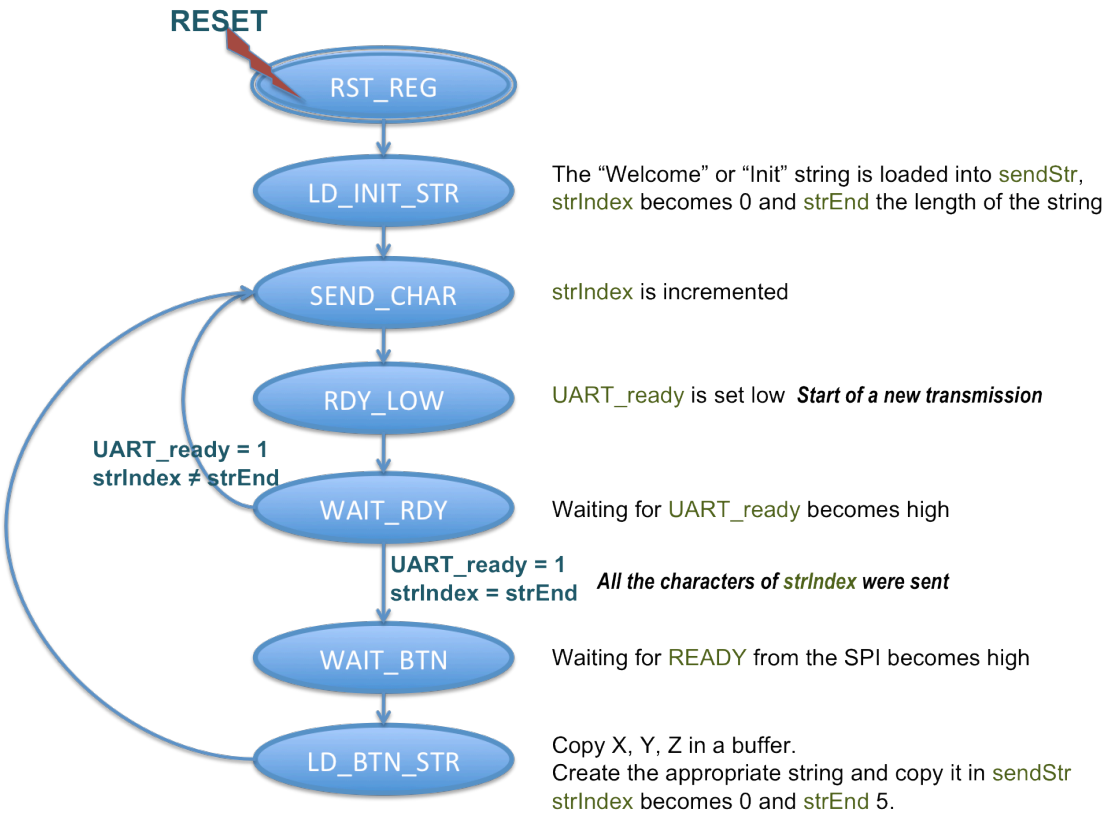
Currently, there is only one sensor and two types of messages (message with data or message with information). Obviously the system will not have 256 sensors or 256 different types of message so I could merge the first two characters and use the 4 MSB for the sensors and the 4 LSB for the type of messages, but the code is easier to understand using ASCII characters.

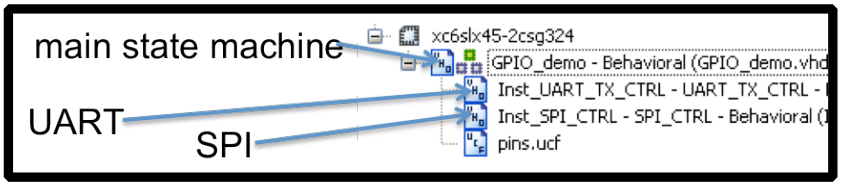
Since it takes less than 7.3µs to acquire data from the 3 axes of the accelerometer and 5ms to send them to the computer, I chose to make the process of sending the data to the computer synchronized with the reception from the accelerometer.



Using this architecture ensures that the value sent by the UART is not written at the same time by the SPI. Moreover, as a SPI cycle is about 680 times shorter than an UART cycle, it is possible to add (for example) an averaging procedure of several values before setting READY to the high state.

Both the SPI and UART are operated in an independent VHDL file. The SPI is actually a continuous counter loop that never stops after reset and the UART is directly controlled by the main state machine that is described below.

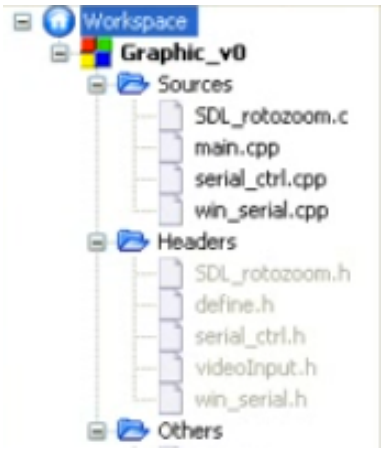




The Graphic interface

To develop the graphic interface I used Code::Blocks with the SDL library and TTF_Font.
SDL (*Simple DirectMedia Layer*) is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer. The advantage of this library, is that, like Code::Blocks, it is a cross-platform. Therefore, it would be relatively simple to make a port for Linux or Mac OS. However, this library only allows low level access. There is no automatic generation of button or any other graphic object. Therefore it is necessary to create, design and manually place each object in the interface.
The objective of this software was to create a simple and user-friendly interface with a representation of all the necessary parameters to make it adaptable. I didn't work on making a realistic representation of any limb since according to the PLP project description; the production-oriented muscle-based human character skinning technique is already very advanced.

The Code :: Blocks project contains many files, each of them performing a particular function:



- **main.cpp** : Contains the main loop in which the screen is displayed and the events (keyboard, mouse) are processed.

- **define.h** : Contains all the constants and especially the ones related to the position of the objects of the graphical interface.

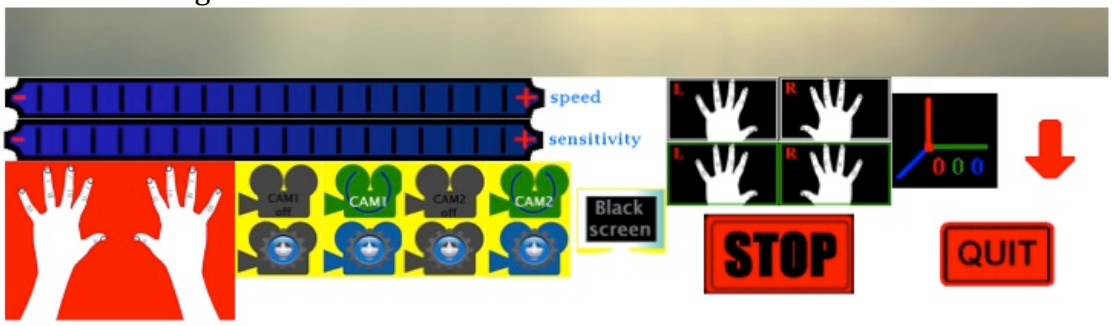
- **win_serial.cpp** : Contains the necessary functions to communicate with the RS232 protocol.

- **win_serial.h** : win_serial.cpp header.

- **serial_ctrl.cpp** : It uses win_serial.cpp and contains more advanced functions to control specifically the FPGA of this project. It finds the right COM port and reads regularly the data (with timer).

- **serial_crtl.h** : serial_ctrl.cpp header.

- All images I drew.



External resources

- **SDL_rotozoom.c** & **SDL_rotozoom.h**: files from the SDL_gfx library containing functions that allows resizing a picture with anti-aliasing.

- **videoInput.h**: header of the VideoInput library that allows taking pictures from a webcam.

C - Create a prototype

C.1 - Difficulties

- The main technical difficulty that I met was the use of a single pin of the FPGA both as an input and output. Indeed, as the SPI was "3 wires", SDI and SDO were on the same pin. However, it is not physically impossible that the same wire is controlled by two different entities and for this reason; it is not possible to get data from the accelerometer without further treatment.

  ➔ The solution is firstly to use an *inout* I/O and secondly to use a signal that I named *oe,* which indicates the direction of the I/O. Based on the *oe* signal, the port inManageout is connected either towards an output or an input that are both internal signals.

```
entity SPI_CTRL is
        Port (  SDIO : inout  STD_LOGIC;
                [...]
        );

architecture Behavioral of SPI_CTRL is
signal SDIO_in          : std_logic;
signal SDIO_out         : std_logic;
signal oe               : std_logic;
[...]

begin
SDIO_in <= SDIO when oe = '0' else '0';      -- oe = '0' => SDIO is an input
SDIO <= SDIO_out when oe = '1' else 'Z';     -- oe = '1' => SDIO is an output
```

- The second challenge was to manage my time. As this is usual in research projects, I was free to work the hours I wanted and move forward in the project according to my desire. However, I only had a limited amount of time (3 months) to complete my internship. Therefore, on the one hand I had to set goals and achieve them as much as possible, and on the other hand, I had to limit my objectives so as not to try developing something that I would not have time to finish. I also had to balance research and development. I spent much time at the beginning of my internship to learn about the internal forward model, but then I did not know how to apply in a practical way what I learned.

C.2 - Make it reusable

This is the second year that this project is made and if it continues next year, it would be of interest that this work be transmitted in the easiest possible way.
For this reason, I committed to put:

- Necessary comments in the code,
- All constants in the file define.h so the source code contains the name of this constants instead of fixed value.
- Useful information to retrieve my work in the "Technical Manual" section.

In addition, my work and the sources I used can be found on the web site plp.antoinem.com with the proper licenses.

C.3 - Improvements

After three months of work, I can look back at my own work and find some possibilities of improvement.

- Currently there is only one sensor, which is good in terms of number of necessary equipment. However, this sensor requires an FPGA as an intermediary between it and the computer. Given the frequency of reading of the accelerometer, a simple microprocessor such as an 8bit PIC would be equally effective and simpler to implement.
- The MMA7455 was slightly noisy and as the position was estimated from the double integral of the acceleration, it carried this initial error. For that reason, I believe that a gyroscope coupled with the accelerometer would provide more accurate results.
- The "videoinput" library I used to capture images with the webcam is not cross-platform and it would be nice to replace it with another library to make the entire project compilable on any system.
- Obviously, the system can still be improved. One could rely a lot more on the equations. I believe it would be possible to develop scenarios inside the HMI where the user is guided through different experiments in order to more accurately calibrate the system. The aim would be to determine factors a, b, k, α in the above mentioned equations with the largest possible accuracy.

Conclusion

The work I've done is a small part of the broader PLP project. The 3 months of my own project were indeed a short time to match my objectives:

- On the one hand, I had to learn a lot in the field of human movement, which is a fascinating new discipline, close to the field of medicine.
- On the other hand, I could use my engineering knowledge to develop software solving the many specific issues needed for a better man machine interface like the SPI and RS232 protocols, the drawing of each element in the interface, etc.

After my short involvement in the PLP project, I can see many opportunities for further developments. I already proposed some practical improvements of the interface. But I also believe it would be valuable to include more knowledge into my software (such as Fitts' law or MacKenzie's low). Finally, I am sure that further progress can only be achieved if the future works rely on interdisciplinary teams mixing both engineering and medical competences.

So of course, what I developed is not yet testable with patients and it is only a glimpse of the many possibilities of the PLP project. But I feel it is a step in the right direction and eventually, the sum of such contributions will help solving the difficult problem of phantom pains experienced by amputees.

Resources

Software and Libraries

- FPGA
  - Xilinx ISE 14.1 webpack - windows XP
  - ATLYS board (Digilent)
  - UART to USB(xr21v1410)
- Accelerometer
  - Parallax MMA7455
  - Freescale MMA7455L (datasheet)

- Graphic interface
  - Development environment : MinGW
  - IDE : Code::Blocks - windows XP
  - Graphics library : SDL
    - SDL_ttf
    - SDL_gfx
  - Capture from webcam : VideoInput

References

- The McGill Pain Questionnaire: Major properties and scoring methods [Ronald Melzack]

- Mémoire d'Habilitation. Modèles d'analyse synthèse de mouvements [Sylvie Gibet]

- Motor Control-Learning Model for Reaching Movements [Hiroyuki Kambara, Kyoungsik Kim, Duk Shin, Makoto Sato, and Yasuharu Koike]

- Fitts' Law and Expanding Targets: An Experimental Study, and Applications to User Interface Design [Michael John MCGuffin]

- http://wikipedia.fr - wikipedia.com

- http://wiki.libsdl.org

- http://www.siteduzero.com/

- http://sdl.beuc.net/

Abbreviations

DDC	Display Data Channel
FPGA	Field-Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphic User Interface
HMI	Human Machine Interface
I^2C	IIC. Inter-Integrated Circuit
LSB	Last Significant Bit
MSB	Most Significant Bit
PCB	Printed Circuit Board
PLP	Phantom Limb Pain
SCL	Serial Clock Line
SDA	Serial Data Line
UART	Universal Asynchronous Receiver and Transmitter
USB	Universal Serial Bus
VGA	Video Graphics Array

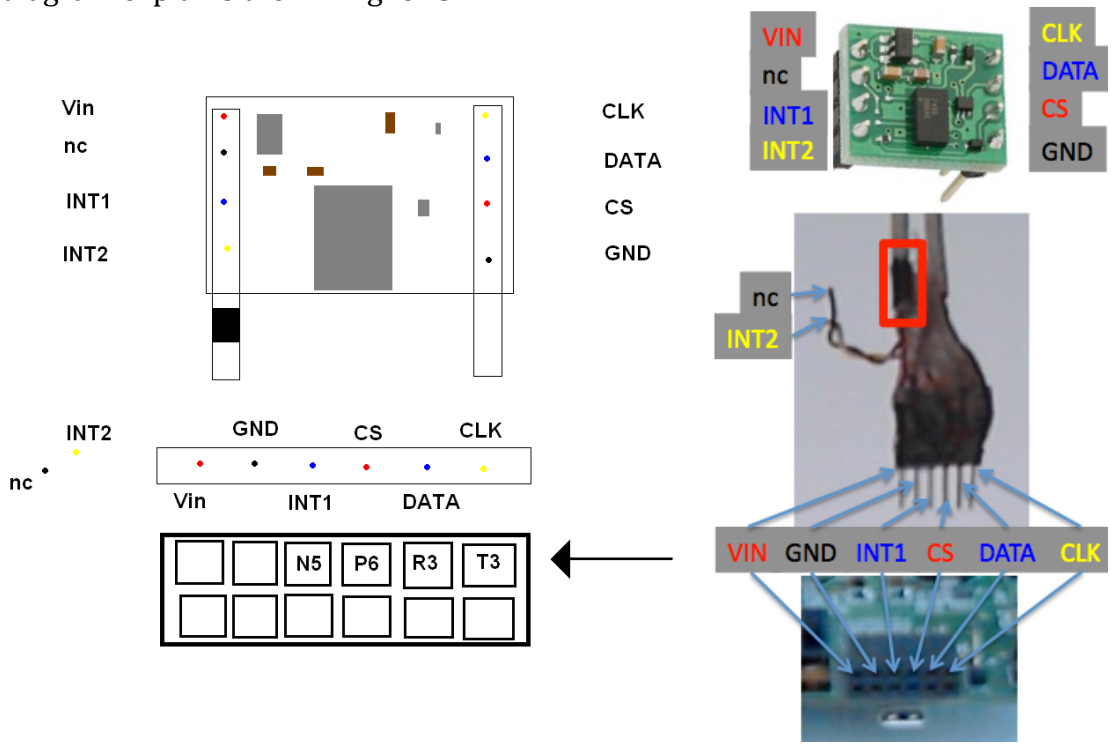
Technical Manual

The purpose of this section is to explain the main characteristics of the source code to enable easier understanding of the code and further reuse of my project. All my developments can be found on website plp.antoinem.com.

FPGA

Wiring

I created two different cables to communicate with the accelerometer. A first one to be used with I2C and a second one with SPI (added SC and INT1). This diagram explains the wiring for SPI.



SPI

The SPI is operated in file “I2C_CTRL.vhd”. This is consistent with the initial developments where the I2C was used to transmit data. The file name was not modified in further developments to avoid errors.
SPI operation is based on counter "cpt" which is controlled by a separate process. During reset, cpt is set to 0 then it is incremented at each clock cycle. Another process sends or reads data depending on the value of cpt. Between 0 and 1365, the initialization commands are sent. Between 1365 and 2717 the value of accelerations on X, Y, Z are read. In 2800, a READY signal is sent to the main state machine to mention new values. Therefore, after initialization, cpt loops from 1365 to read the axes continuously. Since the main state machine is

synchronized to the SPI, it is possible to give a high value to cpt to slow the whole program. For example

```
elsif rising_edge(CLK) then
    if cpt = 1000000 then
        cpt <= 1365;
```

By doing a loop of size 1000000 with a 100MHz clock, the SPI will be read every 10ms, which will slow down the state machine. For example, this can be useful to debug.

- Initialization

During initialization, three registers are set.

$16: Mode Control Register

--	DRPD	SPI3W	STON	GLVL[1]	GLVL[0]	MODE[1]	MODE[0]
0	0	1	0	0	1	0	1

DRPD = 0 : Data ready is output to INT1/DRDY PIN
SPI3W = 1 : SPI is 3 wire mode (SDI and SDO are the same pin)
STON = 0 : Self-test is not enabled
GLVL = 01 : ±2g is selected for measurement range (-2g = $80; +2g = $7F)
MODE = 01 : Measurement mode (continuous measurement)

I choose the 2g-measurement mode because this range is enough for arm movement. And, as I wired the INT1 output on the connector that I made, I set DRPD to 0 in order to output "data ready" on this pin.

$18 Control 1

DFBW	THOPT	ZDA	YDA	XDA	INTREG[1]	INTREG[0]	INTPIN
1	1	1	1	1	0	0	0

DFBW = 1 : Digital filter band width is 125Hz
THOPT = 1 : Integer value is available
ZDA = 1 : Z-axis is disabled for detection
YDA = 1 : Y-axis is disabled for detection
XDA = 1 : X-axis is disabled for detection

$0D: I2C Device Address

I2CDIS	DVAD[6]	DVAD[5]	DVAD[4]	DVAD[3]	DVAD[2]	DVAD[1]	DVAD[0]
1	0	0	1	1	1	0	1

I2CDIS = 1 : I2C is disabled
DVAD = 0011101 : Default I2C device address

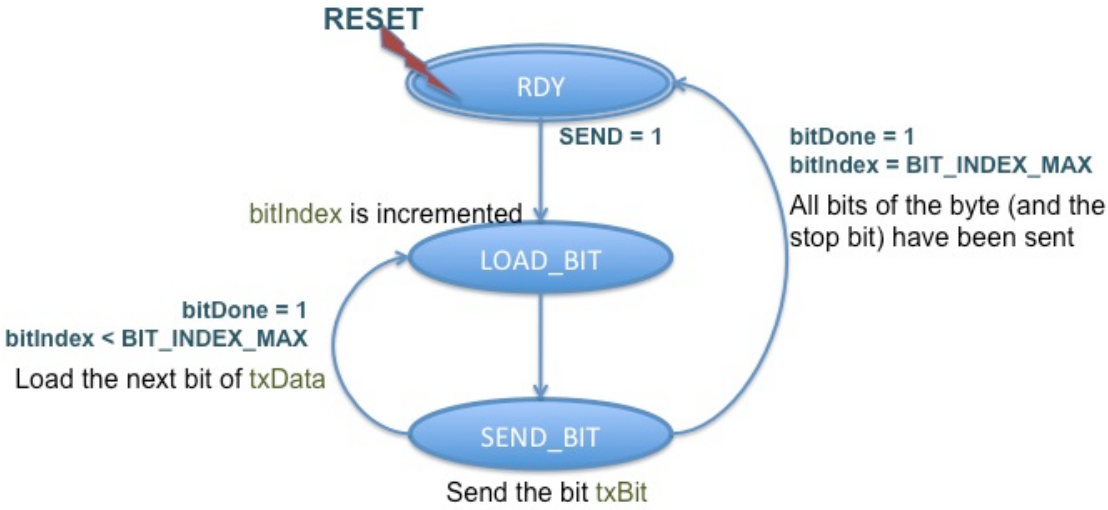
To avoid any conflict between SPI and I2C, I deactivated I2C as recommended by the datasheet.

- Acquisition

When the acquisition takes place, registers XOUT ($06), YOUT ($07) and ZOUT ($08) are read.

UART

The UART is operated in the file "UART_TX_CTRL.vhd". I modified the sample project provided on the Xilinx website to demonstrate usage of the main I/O.



The state machine above can send a single 8-bit byte with a stop bit and no parity. It is controlled by the main state machine programmed in the file "GPIO_demo.vhd" through the SEND signal.

Changing the value of BIT_TMR_MAX can modify the baud rate of the COM port. For a Baud Rate of 9600, BIT_TMR_MAX should be (100Mhz/9600)-1 = 10416 = 0b10100010110000.

HMI

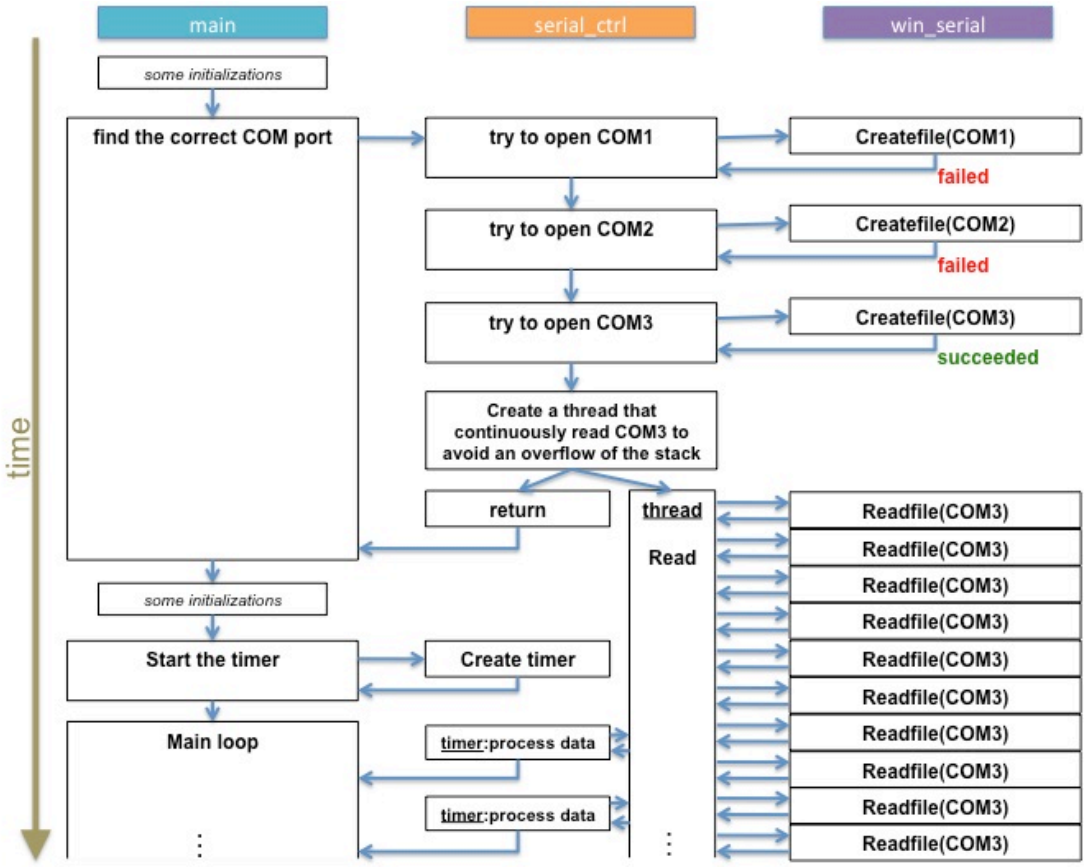
The HMI includes a graphic, an intelligence and a communication part.

RS232

- To open a COM port with windows, use the CreateFile() function with the proper parameters and an object DCB to modify the properties of the COM port (baud rate, parity, etc).
- To read from the COM port, use the ReadFile() function
- And to close the COM port, use the CloseHandle() function.

These functions are specific to Windows and it is necessary to perform some processing before or after using them. For example before reading from the COM port you must check the data size in the stack (See the source code attached). Therefore I have grouped these basic functions in the library win_serial. Thus, one can open, set, read and close the COM port easily with specific functions.

The library serial_ctrl uses win_serial. It contains the functions necessary to find the correct COM port, a thread that continuously reads the RS232 to avoid an overflow of the stack and a timer that periodically processes the data. This library is used by main.cpp



The search of the opened COM port blocks the launch of the interface because it tries to open COM1 to COM16 and then start again to COM1. To avoid a blocking I let the possibility inside the source code to deactivate this research. To do this, simply uncomment the first line of file define.h. **#define PROG_MODE**

When this line is commented, pressing the STOP button stops the search of the FPGA and quits the application.
When this line is uncommented, pressing the STOP button stops the search of the FPGA and displays the interface. Obviously, no data will be read from RS232 but the developer can test the interface without plugging the FPGA.

Graphic

With SDL, the graphical display is done with an infinite loop that clears and refreshes the screen permanently. When refreshing the screen, each object is drawn based on its properties (type, transparency) and its x and y coordinates.

```
SDL_Surface *I_set0 = SDL_LoadBMP("set_to_0.bmp");
//defines a surface which contains the image named set_to_0.bmp
SDL_SetColorKey(I_set0, SDL_SRCCOLORKEY, SDL_MapRGB(I_set0->format, 0, 0, 0));
//defines the black (R:0, G:0, B:0) as a transparent color in this picture.
SDL_Rect P_set0;
//defines a rectangle in which the surface will be placed.
P_set0.x = R_P_set0_x;
P_set0.y = R_P_set0_y;
//defines the position x and y of this rectangle (and therefore of the image).
```

```
while (!done)
//beginning of the infinite loop
{
    [...] //some process
    /* Beginning of the screen update */
     SDL_FillRect(screen, 0, SDL_MapRGB(screen->format, 0, 0, 0));
    //clear screen
    SDL_BlitSurface(I_set0, NULL, screen, &P_set0);
    //draw the screen
    SDL_Flip(screen);
    //update the screen
}
```

For example, to move the hand we must increment or decrement the x coordinate and to change the color of a button, it is actually necessary to replace it by another surface containing a different image.

Event

Events (keyboard, mouse) are placed in a queue that we must read continually to treat these events. It is actually done with the function SDL_PollEvent(SDL_Event *event) which stores in event the oldest pending events. Therefore, before refreshing the screen we must call this function and process all pending events.

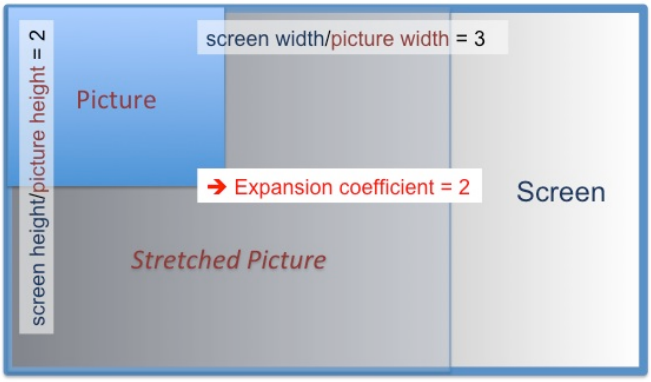
```
while (!done)
{
    SDL_Event event;
    // message processing loop
    while (SDL_PollEvent(&event))
    {
        switch (event.type) // check for messages
        {
            case SDL_QUIT: { // exit if the window is closed
                done = true;
                break;
                }
            case SDL_KEYDOWN: { // check for keypresses
                [...] //some process
                break;
                }
            case SDL_MOUSEMOTION: //check for mouse motion
                [...] //some process
                break;
                }
            Others case
        } // end switch
    } // end of message processing
/* SCREEN UPDATE */
} // --- END MAIN LOOP ---
```

Take picture

To take a picture I use the library videoInput, which allows detecting the webcams plugged on the computer and to fill a buffer with raw data extracted from this camera. First of all, an object of type videoInput is needed. Then it is possible to print out a list of available devices and returns the number of devices found. We can then set each camera independently: frame rate, connection type, format, and video size before starting to record. The data are placed into a buffer which is passed as a parameter of the function getPixels().
However these are raw data and to use them you must know the number of bits per pixel, the height and width of the image to reconstruct an picture from the buffer which have only one dimension. We call the function getWidth(device), getHeight(device) and getSize(device) and it is possible to determine the number of bits per pixel $bpp = \frac{size}{width * height} \times 8$. For example if bpp is 32, it is necessary to use 4 bytes for each pixel: so the size of the buffer will be 4*(number of pixel in the picture). And because the number of pixel in the picture is (width*height), we find that bpp=4*8.
Once, we know all this information, it is possible to use the function SDL_CreateRGBSurfaceFrom() to create a Surface from the buffer.

```
SDL_Surface *SDL_CreateRGBSurfaceFrom(void *pixels, int width, int height, int depth, int pitch, Uint32 Rmask, Uint32 Gmask, Uint32 Bmask, Uint32 Amask)
```

```
I_background = SDL_CreateRGBSurfaceFrom(buffer, width, height, (size/(height*width))*8, size/camheight, 0, 0, 0, 0)
```



Unfortunately, the dimensions of this surface correspond to the dimensions of the image taken from the webcam and we must stretch the image to put it in the background. This requires finding the smallest ratio between (screen width/picture width) and (screen height/picture height) to use it as expansion coefficient.

To stretch the picture properly I used the library SDL_gfx, which contains the files rotozoom.c and rotozoom.h. Their functions allow resizing a picture with appropriate algorithms like anti-aliasing.

```
SDL_Surface *zoomSurface(SDL_Surface * src, double zoomx, double zoomy, int smooth)
```

```
----------------------------------------------------------------------------------
--   GPIO_demo.vhd -- Atlys GPIO/UART Demonstration Project
----------------------------------------------------------------------------------
-- Author:  Sam Bobrowicz
--          Copyright 2011 Digilent, Inc.
----------------------------------------------------------------------------------
--
----------------------------------------------------------------------------------
--   The GPIO/UART Demo project demonstrates a simple usage of the Atlys's
-- GPIO and UART in an ISE design. The behavior is as follows:
--
--         *The 8 User LEDs are tied to the 8 User Switches. While the center
--             User button is pressed, the LEDs are instead tied to GND
--        *An introduction message is sent across the UART when the device
--         is finished being configured, and after the center User button
--         is pressed.
--        *A message is sent over UART whenever BTNU, BTNL, BTND, or BTNR is
--         pressed.
--        *Note that the center user button behaves as a user reset button
--         and is referred to as such in the code comments below
--
--   All UART communication can be captured by attaching the UART port to a
-- computer running a Terminal program with 230400 Baud Rate, 8 data bits, no
-- parity, and 1 stop bit.

----------------------------------------------------------------------------------
--
----------------------------------------------------------------------------------
-- Revision History:
--  11/3/2011(SamB): Created using Xilinx Tools 13.2
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--The IEEE.std_logic_unsigned contains definitions that allow
--std_logic_vector types to be used with the + operator to instantiate a
--counter.
use IEEE.std_logic_unsigned.all;

entity GPIO_demo is
    Port ( CLK : in  STD_LOGIC;
                RESET : in STD_LOGIC;
           UART_TXD : out  STD_LOGIC;
                --SPI
                SC : out  STD_LOGIC;
           SDIO : inout  STD_LOGIC;
           SPC : out  STD_LOGIC;
                --IO
                LED : out std_logic_vector(7 downto 0);
                BTN : in std_logic_vector(2 downto 0)
              );
end GPIO_demo;

architecture Behavioral of GPIO_demo is

component UART_TX_CTRL
Port(
     SEND : in std_logic;
     DATA : in std_logic_vector(7 downto 0);
     CLK : in std_logic;
     READY : out std_logic;
     UART_TX : out std_logic
     );
```

```
end component;

component SPI_CTRL
Port(
     CLK : in  STD_LOGIC;
   RESET : in  STD_LOGIC;
   SC : out  STD_LOGIC;
   SDIO : inout  STD_LOGIC;
   SPC : out  STD_LOGIC;
     XOUT : out std_logic_vector(7 downto 0);
     YOUT : out std_logic_vector(7 downto 0);
     ZOUT : out std_logic_vector(7 downto 0);
     READY : out std_logic
     );
end component;

--The type definition for the UART state machine type. Here is a description of what
--occurs during each state:
-- RST_REG     -- Do Nothing. This state is entered after configuration or a user reset.
--                The state is set to LD_INIT_STR.
-- LD_INIT_STR -- The Welcome String is loaded into the sendStr variable and the strIndex
--                variable is set to zero. The welcome string length is stored in the
StrEnd
--                variable. The state is set to SEND_CHAR.
-- SEND_CHAR   -- uartSend is set high for a single clock cycle, signaling the character
--                data at sendStr(strIndex) to be registered by the UART_TX_CTRL at the
next
--                cycle. Also, strIndex is incremented (behaves as if it were post
--                incremented after reading the sendStr data). The state is set to
RDY_LOW.
-- RDY_LOW     -- Do nothing. Wait for the READY signal from the UART_TX_CTRL to go low,
--                indicating a send operation has begun. State is set to WAIT_RDY.
-- WAIT_RDY    -- Do nothing. Wait for the READY signal from the UART_TX_CTRL to go high,
--                indicating a send operation has finished. If READY is high and strEnd =
--                StrIndex then state is set to WAIT_BTN, else if READY is high and
strEnd /=
--                StrIndex then state is set to SEND_CHAR.
-- WAIT_BTN    -- Do nothing. Wait for a button press on BTNU, BTNL, BTND, or BTNR. If a
--                button press is detected, set the state to LD_BTN_STR.
-- LD_BTN_STR  -- The Button String is loaded into the sendStr variable and the strIndex
--                variable is set to zero. The button string length is stored in the
StrEnd
--                variable. The state is set to SEND_CHAR.
type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN,
LD_BTN_STR);

--The CHAR_ARRAY type is a variable length array of 8 bit std_logic_vectors.
--Each std_logic_vector contains an ASCII value and represents a character in
--a string. The character at index 0 is meant to represent the first
--character of the string, the character at index 1 is meant to represent the
--second character of the string, and so on.
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

constant STR_LEN : integer := 5;

--Welcome string definition. Note that the values stored at each index
--are the ASCII values of the indicated character.
constant WELCOME_STR : CHAR_ARRAY(0 to 4) :=  (X"41",  --A as in 'this is thaaaaaa
correct COM port'

                X"20",  --space as in nothing

                X"4F",  --O
```

```
                X"4B",  --K

                X"20"); --space


--Button press string definition.
constant BTN_STR : CHAR_ARRAY(0 to 4) :=     (X"41",   --A as in 'this is thaaaaaa
correct COM port'

                X"64",   --d as in data

                X"58",   --X

                X"59",   --Y

                X"5A");  --Z

--Contains the current string being sent over uart.
signal sendStr : CHAR_ARRAY(0 to (STR_LEN - 1));

--Contains the length of the current string being sent over uart.
signal strEnd : natural;

--Contains the index of the next character to be sent over uart
--within the sendStr variable.
signal strIndex : natural;

--UART_TX_CTRL control signals
signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

--Current uart state signal
signal uartState : UART_STATE_TYPE := RST_REG;

--SPI signals
signal spi_x : std_logic_vector(7 downto 0);
signal buf_spi_x : std_logic_vector(7 downto 0);
signal spi_y : std_logic_vector(7 downto 0);
signal buf_spi_y : std_logic_vector(7 downto 0);
signal spi_z : std_logic_vector(7 downto 0);
signal buf_spi_z : std_logic_vector(7 downto 0);
signal spi_ready : std_logic;

begin

--print xyz on LED
process(BTN)
begin
     if BTN = "001" then
          LED <= buf_spi_x;
     else
          if BTN = "010" then
               LED <= buf_spi_y;
          else
               if BTN = "100" then
                    LED <= buf_spi_z;
               else
                    LED <= (others => '0');
               end if;
          end if;
     end if;
end process;
```

```
--Next Uart state logic (states described above)
next_uartState_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          if (RESET = '1') then
               uartState <= RST_REG;
               buf_spi_x <= (others => '0');
               buf_spi_y <= (others => '0');
               buf_spi_z <= (others => '0');
          else
               case uartState is
               when RST_REG =>
                    uartState <= LD_INIT_STR;
               when LD_INIT_STR =>
                    uartState <= SEND_CHAR;
               when SEND_CHAR =>
                    uartState <= RDY_LOW;
               when RDY_LOW =>
                    uartState <= WAIT_RDY;
               when WAIT_RDY =>
                    if (uartRdy = '1') then
                         if (strEnd = strIndex) then
                              uartState <= WAIT_BTN;
                         else
                              uartState <= SEND_CHAR;
                         end if;
                    end if;
               when WAIT_BTN =>
                    if (spi_ready = '1') then
                         buf_spi_x <= spi_x;
                         buf_spi_y <= spi_y;
                         buf_spi_z <= spi_z;
                         uartState <= LD_BTN_STR;
                    end if;
               when LD_BTN_STR =>
                    uartState <= SEND_CHAR;
               when others=> --should never be reached
                    uartState <= RST_REG;
               end case;
          end if ;
     end if;
end process;

--Loads the sendStr and strEnd signals when a LD state is
--is reached.
string_load_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          if (uartState = LD_INIT_STR) then
               sendStr <= WELCOME_STR;
               strEnd <= STR_LEN;
          elsif (uartState = LD_BTN_STR) then
               sendStr <= (X"41", X"64", buf_spi_x, buf_spi_y, buf_spi_z);
               strEnd <= STR_LEN;
          end if;
     end if;
end process;

--Conrols the strIndex signal so that it contains the index
--of the next character that needs to be sent over uart
char_count_process : process (CLK)
begin
     if (rising_edge(CLK)) then
```

```
          if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
               strIndex <= 0;
          elsif (uartState = SEND_CHAR) then
               strIndex <= strIndex + 1;
          end if;
     end if;
end process;

--Controls the UART_TX_CTRL signals
char_load_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          if (uartState = SEND_CHAR) then
               uartSend <= '1';
               uartData <= sendStr(strIndex);
          else
               uartSend <= '0';
          end if;
     end if;
end process;

--Component used to send a byte of data over a UART line.
Inst_UART_TX_CTRL: UART_TX_CTRL port map(
          SEND => uartSend,
          DATA => uartData,
          CLK => CLK,
          READY => uartRdy,
          UART_TX => uartTX
     );

UART_TXD <= uartTX;

Inst_SPI_CTRL: SPI_CTRL port map(
          CLK => CLK,
       RESET => RESET,
       SC => SC,
       SDIO => SDIO,
       SPC => SPC,
          XOUT => spi_x,
          YOUT => spi_y,
          ZOUT => spi_z,
          READY => spi_ready
     );

end Behavioral;
```

```
----------------------------------------------------------------------------------
--   UART_TX_CTRL.vhd -- UART Data Transfer Component
----------------------------------------------------------------------------------
-- Author:  Sam Bobrowicz
--          Copyright 2011 Digilent, Inc.
----------------------------------------------------------------------------------
--
----------------------------------------------------------------------------------
--   This component may be used to transfer data over a UART device. It will
-- serialize a byte of data and transmit it over a TXD line. The serialized
-- data has the following characteristics:
--         *230400 Baud Rate
--         *8 data bits, LSB first
--         *1 stop bit
--         *no parity
--
-- Port Descriptions:
--
--    SEND - Used to trigger a send operation. The upper layer logic should
--           set this signal high for a single clock cycle to trigger a
--           send. When this signal is set high DATA must be valid . Should
--           not be asserted unless READY is high.
--    DATA - The parallel data to be sent. Must be valid the clock cycle
--           that SEND has gone high.
--    CLK  - A 100 MHz clock is expected
--   READY - This signal goes low once a send operation has begun and
--           remains low until it has completed and the module is ready to
--           send another byte.
-- UART_TX - This signal should be routed to the appropriate TX pin of the
--           external UART device.
--
----------------------------------------------------------------------------------
--
----------------------------------------------------------------------------------
-- Revision History:
--  08/08/2011(SamB): Created using Xilinx Tools 13.2
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

--constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "00000110110001"; --433 =
(round(100MHz / 230400)) - 1
constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000"; --10416 =
(round(100MHz / 9600)) - 1
constant BIT_INDEX_MAX : natural := 10;

--Counter that keeps track of the number of clock cycles the current bit has been held
stable over the
--UART TX line. It is used to signal when the ne
signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');
```

```
--combinatorial logic that goes high when bitTmr has counted to the proper value to
ensure
--a 9600 baud rate
signal bitDone : std_logic;

--Contains the index of the next bit in txData that needs to be transferred
signal bitIndex : natural;

--a register that holds the current data being sent over the UART TX line
signal txBit : std_logic := '1';

--A register that contains the whole data packet to be sent, including start and stop
bits.
signal txData : std_logic_vector(9 downto 0);

signal txState : TX_STATE_TYPE := RDY;

begin

--Next state logic
next_txState_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          case txState is
          when RDY =>
               if (SEND = '1') then
                    txState <= LOAD_BIT;
               end if;
          when LOAD_BIT =>
               txState <= SEND_BIT;
          when SEND_BIT =>
               if (bitDone = '1') then
                    if (bitIndex = BIT_INDEX_MAX) then
                         txState <= RDY;
                    else
                         txState <= LOAD_BIT;
                    end if;
               end if;
          when others=> --should never be reached
               txState <= RDY;
          end case;
     end if;
end process;

bit_timing_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          if (txState = RDY) then
               bitTmr <= (others => '0');
          else
               if (bitDone = '1') then
                    bitTmr <= (others => '0');
               else
                    bitTmr <= bitTmr + 1;
               end if;
          end if;
     end if;
end process;

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else
                    '0';

bit_counting_process : process (CLK)
begin
```

```
     if (rising_edge(CLK)) then
          if (txState = RDY) then
               bitIndex <= 0;
          elsif (txState = LOAD_BIT) then
               bitIndex <= bitIndex + 1;
          end if;
     end if;
end process;

tx_data_latch_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          if (SEND = '1') then
               txData <= '1' & DATA & '0';
          end if;
     end if;
end process;

tx_bit_process : process (CLK)
begin
     if (rising_edge(CLK)) then
          if (txState = RDY) then
               txBit <= '1';
          elsif (txState = LOAD_BIT) then
               txBit <= txData(bitIndex);
          end if;
     end if;
end process;

UART_TX <= txBit;
READY <= '1' when (txState = RDY) else
               '0';

end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    22:00:04 06/28/2012
-- Design Name:
-- Module Name:    I2C_CTRL - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SPI_CTRL is
    Port ( CLK : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
--           INT1 : in  STD_LOGIC;
           SC : out  STD_LOGIC;
           SDIO : inout  STD_LOGIC;
           SPC : out  STD_LOGIC;
                XOUT : out std_logic_vector(7 downto 0);
                YOUT : out std_logic_vector(7 downto 0);
                ZOUT : out std_logic_vector(7 downto 0);
                READY : out std_logic
              );
end SPI_CTRL;

architecture Behavioral of SPI_CTRL is

-- signal
signal cpt                      : integer RANGE 0 to 200000;
signal acc_x                : std_logic_vector(7 downto 0);
signal acc_y                : std_logic_vector(7 downto 0);
signal acc_z                : std_logic_vector(7 downto 0);
signal SDIO_in                  : std_logic;
signal SDIO_out             : std_logic;
signal oe                   : std_logic;

-- constant
constant ADD_MODE_CTRL  : std_logic_vector(7 downto 0) := "10101100"; -- 1:W -- 010110:x16
-- 0:don't care
constant DATA_MODE_CTRL : std_logic_vector(7 downto 0) := "00100101";
constant ADD_CTRL1          : std_logic_vector(7 downto 0) := "10110000"; -- 1:W --
011000:x18 -- 0:don't care
constant DATA_CTRL1         : std_logic_vector(7 downto 0) := "11111000";
constant ADD_I2C_DEV    : std_logic_vector(7 downto 0) := "10011010"; -- 1:W -- 001101:x0D
```

```
-- 0:don't care
constant DATA_I2C_DEV   : std_logic_vector(7 downto 0) := "10011101";
constant ADD_XOUT       : std_logic_vector(7 downto 0) := "00001100"; -- 0:R -- 000110:x06
-- 0:don't care
constant ADD_YOUT       : std_logic_vector(7 downto 0) := "00001110"; -- 0:R -- 000111:x07
-- 0:don't care
constant ADD_ZOUT       : std_logic_vector(7 downto 0) := "00010000"; -- 0:R -- 001000:x08
-- 0:don't care

begin

process(CLK, RESET)
begin
     if RESET = '1' then
          cpt <= 0;
     elsif rising_edge(CLK) then
          if cpt = 100000 then
               cpt <= 1365;
          else
               cpt<= cpt+1;
          end if;
     end if;
end process;


SDIO_in <= SDIO when oe = '0' else '0';         -- oe = '0' => SDIO is an input
SDIO <= SDIO_out when oe = '1' else 'Z';        -- oe = '1' => SDIO is an output

process(CLK, RESET)     -- continous SPC
begin
     if RESET = '1' then
          SC <= '1';
          SPC <= '0';
          oe <= '1';
          XOUT <= (others => '0');
          YOUT <= (others => '0');
          ZOUT <= (others => '0');
          READY <= '0';
     elsif rising_edge(clk) then
          case cpt is
               when 0 =>
                    SC <='0';                                   -- DEBUT DE LA TRANSMISSION 1
                    SDIO_out <= ADD_MODE_CTRL(7);
                    SPC <= '0';
               when 13 =>
                    SPC <= '1';
               when 26 =>
                    SDIO_out <= ADD_MODE_CTRL(6);
                    SPC <= '0';
               when 39 =>
                    SPC <= '1';
               when 52 =>
                    SDIO_out <= ADD_MODE_CTRL(5);
                    SPC <= '0';
               when 65 =>
                    SPC <= '1';
               when 78 =>
                    SDIO_out <= ADD_MODE_CTRL(4);
                    SPC <= '0';
               when 91 =>
                    SPC <= '1';
               when 104 =>
                    SDIO_out <= ADD_MODE_CTRL(3);
                    SPC <= '0';
```

```
            when 117 =>
                 SPC <= '1';
            when 130 =>
                 SDIO_out <= ADD_MODE_CTRL(2);
                 SPC <= '0';
            when 143 =>
                 SPC <= '1';
            when 156 =>
                 SDIO_out <= ADD_MODE_CTRL(1);
                 SPC <= '0';
            when 169 =>
                 SPC <= '1';
            when 182 =>
                 SDIO_out <= ADD_MODE_CTRL(0);
                 SPC <= '0';
            when 195 =>
                 SPC <= '1';
            when 208 =>
                 SDIO_out <= DATA_MODE_CTRL(7);
                 SPC <= '0';
            when 221 =>
                 SPC <= '1';
            when 234 =>
                 SDIO_out <= DATA_MODE_CTRL(6);
                 SPC <= '0';
            when 247 =>
                 SPC <= '1';
            when 260 =>
                 SDIO_out <= DATA_MODE_CTRL(5);
                 SPC <= '0';
            when 273 =>
                 SPC <= '1';
            when 286 =>
                 SDIO_out <= DATA_MODE_CTRL(4);
                 SPC <= '0';
            when 299 =>
                 SPC <= '1';
            when 312 =>
                 SDIO_out <= DATA_MODE_CTRL(3);
                 SPC <= '0';
            when 325 =>
                 SPC <= '1';
            when 338 =>
                 SDIO_out <= DATA_MODE_CTRL(2);
                 SPC <= '0';
            when 351 =>
                 SPC <= '1';
            when 364 =>
                 SDIO_out <= DATA_MODE_CTRL(1);
                 SPC <= '0';
            when 377 =>
                 SPC <= '1';
            when 390 =>
                 SDIO_out <= DATA_MODE_CTRL(0);
                 SPC <= '0';
            when 403 =>
                 SPC <= '1';
            when 416 =>
                 SPC <= '0';
            when 429 =>
                 SC <= '1';
    -- FIN DE LA TRANSMISSION 1
            when 455 =>
                 SC <= '0';
```

```
	-- DEBUT DE LA TRANSMISSION 2
				SDIO_out <= ADD_CTRL1(7);
			when 468 =>
				SPC <= '1';
			when 494 =>
				SDIO_out <= ADD_CTRL1(6);
				SPC <= '0';
			when 507 =>
				SPC <= '1';
			when 520 =>
				SDIO_out <= ADD_CTRL1(5);
				SPC <= '0';
			when 533 =>
				SPC <= '1';
			when 546 =>
				SDIO_out <= ADD_CTRL1(4);
				SPC <= '0';
			when 559 =>
				SPC <= '1';
			when 572 =>
				SDIO_out <= ADD_CTRL1(3);
				SPC <= '0';
			when 585 =>
				SPC <= '1';
			when 598 =>
				SDIO_out <= ADD_CTRL1(2);
				SPC <= '0';
			when 611 =>
				SPC <= '1';
			when 624 =>
				SDIO_out <= ADD_CTRL1(1);
				SPC <= '0';
			when 637 =>
				SPC <= '1';
			when 650 =>
				SDIO_out <= ADD_CTRL1(0);
				SPC <= '0';
			when 663 =>
				SPC <= '1';
			when 676 =>
				SDIO_out <= DATA_CTRL1(7);
				SPC <= '0';
			when 689 =>
				SPC <= '1';
			when 702 =>
				SDIO_out <= DATA_CTRL1(6);
				SPC <= '0';
			when 715 =>
				SPC <= '1';
			when 728 =>
				SDIO_out <= DATA_CTRL1(5);
				SPC <= '0';
			when 741 =>
				SPC <= '1';
			when 754 =>
				SDIO_out <= DATA_CTRL1(4);
				SPC <= '0';
			when 767 =>
				SPC <= '1';
			when 780 =>
				SDIO_out <= DATA_CTRL1(3);
				SPC <= '0';
			when 793 =>
				SPC <= '1';
```

```
            when 806 =>
                 SDIO_out <= DATA_CTRL1(2);
                 SPC <= '0';
            when 819 =>
                 SPC <= '1';
            when 832 =>
                 SDIO_out <= DATA_CTRL1(1);
                 SPC <= '0';
            when 845 =>
                 SPC <= '1';
            when 858 =>
                 SDIO_out <= DATA_CTRL1(0);
                 SPC <= '0';
            when 871 =>
                 SPC <= '1';
            when 884 =>
                 SPC <= '0';
            when 897 =>
                 SC <= '1';
    -- FIN DE LA TRANSMISSION 2
            when 923 =>
                 SC <= '0';
    -- DEBUT DE LA TRANSMISSION 3
                 SDIO_out <= ADD_I2C_DEV(7);
            when 936 =>
                 SPC <= '1';
            when 949 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(6);
            when 962 =>
                 SPC <= '1';
            when 975 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(5);
            when 988 =>
                 SPC <= '1';
            when 1001 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(4);
            when 1014 =>
                 SPC <= '1';
            when 1027 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(3);
            when 1040 =>
                 SPC <= '1';
            when 1053 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(2);
            when 1066 =>
                 SPC <= '1';
            when 1079 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(1);
            when 1092 =>
                 SPC <= '1';
            when 1105 =>
                 SPC <= '0';
                 SDIO_out <= ADD_I2C_DEV(0);
            when 1118 =>
                 SPC <= '1';
            when 1131 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(7);
```

```
            when 1144 =>
                 SPC <= '1';
            when 1157 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(6);
            when 1170 =>
                 SPC <= '1';
            when 1183 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(5);
            when 1196 =>
                 SPC <= '1';
            when 1209 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(4);
            when 1222 =>
                 SPC <= '1';
            when 1235 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(3);
            when 1248 =>
                 SPC <= '1';
            when 1261 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(2);
            when 1274 =>
                 SPC <= '1';
            when 1287 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(1);
            when 1300 =>
                 SPC <= '1';
            when 1313 =>
                 SPC <= '0';
                 SDIO_out <= DATA_I2C_DEV(0);
            when 1326 =>
                 SPC <= '1';
            when 1339 =>
                 SPC <= '0';
            when 1352 =>
                 SC <= '1';
    -- FIN DE LA TRANSMISSION 3
            when 1365 =>
                 oe <= '1';
    -- we change SDIO to output
            when 1378 =>
                 SC <= '0';
    -- DEBUT DE LA TRANSMISSION 4
                 SDIO_out <= ADD_XOUT(7);
            when 1391 =>
                 SPC <= '1';
            when 1404 =>
                 SPC <= '0';
                 SDIO_out <= ADD_XOUT(6);
            when 1417 =>
                 SPC <= '1';
            when 1430 =>
                 SPC <= '0';
                 SDIO_out <= ADD_XOUT(5);
            when 1443 =>
                 SPC <= '1';
            when 1456 =>
                 SPC <= '0';
                 SDIO_out <= ADD_XOUT(4);
```

```
                when 1469 =>
                     SPC <= '1';
                when 1482 =>
                     SPC <= '0';
                     SDIO_out <= ADD_XOUT(3);
                when 1495 =>
                     SPC <= '1';
                when 1508 =>
                     SPC <= '0';
                     SDIO_out <= ADD_XOUT(2);
                when 1521 =>
                     SPC <= '1';
                when 1534 =>
                     SPC <= '0';
                     SDIO_out <= ADD_XOUT(1);
                when 1547 =>
                     SPC <= '1';
                when 1560 =>
                     SPC <= '0';
                     SDIO_out <= ADD_XOUT(0);
                when 1573 =>
                     SPC <= '1';
                     oe <= '0';                              -- we change SDIO to input
                when 1586 =>
                     SPC <= '0';
                when 1599 =>
                     SPC <= '1';
                     acc_x(7) <= SDIO_in;
                when 1612 =>
                     SPC <= '0';
                when 1625 =>
                     SPC <= '1';
                     acc_x(6) <= SDIO_in;
                when 1638 =>
                     SPC <= '0';
                when 1651 =>
                     SPC <= '1';
                     acc_x(5) <= SDIO_in;
                when 1664 =>
                     SPC <= '0';
                when 1677 =>
                     SPC <= '1';
                     acc_x(4) <= SDIO_in;
                when 1690 =>
                     SPC <= '0';
                when 1703 =>
                     SPC <= '1';
                     acc_x(3) <= SDIO_in;
                when 1716 =>
                     SPC <= '0';
                when 1729 =>
                     SPC <= '1';
                     acc_x(2) <= SDIO_in;
                when 1742 =>
                     SPC <= '0';
                when 1755 =>
                     SPC <= '1';
                     acc_x(1) <= SDIO_in;
                when 1768 =>
                     SPC <= '0';
                when 1781 =>
                     SPC <= '1';
                     acc_x(0) <= SDIO_in;
                when 1794 =>
```

```
                SPC <= '0';
            when 1807 =>
                SC <= '1';
    -- FIN DE LA TRANSMISSION 4
                XOUT <= acc_x;
            when 1820 =>
                oe <= '1';
    -- we change SDIO to output
            when 1833 =>
                SC <= '0';
    -- DEBUT DE LA TRANSMISSION 5
                SDIO_out <= ADD_YOUT(7);
            when 1846 =>
                SPC <= '1';
            when 1859 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(6);
            when 1872 =>
                SPC <= '1';
            when 1885 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(5);
            when 1898 =>
                SPC <= '1';
            when 1911 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(4);
            when 1924 =>
                SPC <= '1';
            when 1937 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(3);
            when 1950 =>
                SPC <= '1';
            when 1963 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(2);
            when 1976 =>
                SPC <= '1';
            when 1989 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(1);
            when 2002 =>
                SPC <= '1';
            when 2015 =>
                SPC <= '0';
                SDIO_out <= ADD_YOUT(0);
            when 2028 =>
                SPC <= '1';
                oe <= '0';                              -- we change SDIO to input
            when 2041 =>
                SPC <= '0';
            when 2054 =>
                SPC <= '1';
                acc_y(7) <= SDIO_in;
            when 2067 =>
                SPC <= '0';
            when 2080 =>
                SPC <= '1';
                acc_y(6) <= SDIO_in;
            when 2093 =>
                SPC <= '0';
            when 2106 =>
                SPC <= '1';
```

```
                acc_y(5) <= SDIO_in;
            when 2119 =>
                SPC <= '0';
            when 2132 =>
                SPC <= '1';
                acc_y(4) <= SDIO_in;
            when 2145 =>
                SPC <= '0';
            when 2158 =>
                SPC <= '1';
                acc_y(3) <= SDIO_in;
            when 2171 =>
                SPC <= '0';
            when 2184 =>
                SPC <= '1';
                acc_y(2) <= SDIO_in;
            when 2197 =>
                SPC <= '0';
            when 2210 =>
                SPC <= '1';
                acc_y(1) <= SDIO_in;
            when 2223 =>
                SPC <= '0';
            when 2236 =>
                SPC <= '1';
                acc_y(0) <= SDIO_in;
            when 2249 =>
                SPC <= '0';
            when 2262 =>
                SC <= '1';
    -- FIN DE LA TRANSMISSION 5
                YOUT <= acc_y;
            when 2275 =>
                oe <= '1';
    -- we change SDIO to output
            when 2288 =>
                SC <= '0';
    -- DEBUT DE LA TRANSMISSION 6
                SDIO_out <= ADD_ZOUT(7);
            when 2301 =>
                SPC <= '1';
            when 2314 =>
                SPC <= '0';
                SDIO_out <= ADD_ZOUT(6);
            when 2327 =>
                SPC <= '1';
            when 2340 =>
                SPC <= '0';
                SDIO_out <= ADD_ZOUT(5);
            when 2353 =>
                SPC <= '1';
            when 2366 =>
                SPC <= '0';
                SDIO_out <= ADD_ZOUT(4);
            when 2379 =>
                SPC <= '1';
            when 2392 =>
                SPC <= '0';
                SDIO_out <= ADD_ZOUT(3);
            when 2405 =>
                SPC <= '1';
            when 2418 =>
                SPC <= '0';
                SDIO_out <= ADD_ZOUT(2);
```

```
            when 2431 =>
                 SPC <= '1';
            when 2444 =>
                 SPC <= '0';
                 SDIO_out <= ADD_ZOUT(1);
            when 2457 =>
                 SPC <= '1';
            when 2470 =>
                 SPC <= '0';
                 SDIO_out <= ADD_ZOUT(0);
            when 2483 =>
                 SPC <= '1';
                 oe <= '0';                              -- we change SDIO to input
            when 2496 =>
                 SPC <= '0';
            when 2509 =>
                 SPC <= '1';
                 acc_z(7) <= SDIO_in;
            when 2522 =>
                 SPC <= '0';
            when 2535 =>
                 SPC <= '1';
                 acc_z(6) <= SDIO_in;
            when 2548 =>
                 SPC <= '0';
            when 2561 =>
                 SPC <= '1';
                 acc_z(5) <= SDIO_in;
            when 2574 =>
                 SPC <= '0';
            when 2587 =>
                 SPC <= '1';
                 acc_z(4) <= SDIO_in;
            when 2600 =>
                 SPC <= '0';
            when 2613 =>
                 SPC <= '1';
                 acc_z(3) <= SDIO_in;
            when 2626 =>
                 SPC <= '0';
            when 2639 =>
                 SPC <= '1';
                 acc_z(2) <= SDIO_in;
            when 2652 =>
                 SPC <= '0';
            when 2665 =>
                 SPC <= '1';
                 acc_z(1) <= SDIO_in;
            when 2678 =>
                 SPC <= '0';
            when 2691 =>
                 SPC <= '1';
                 acc_z(0) <= SDIO_in;
            when 2704 =>
                 SPC <= '0';
            when 2717 =>
                 SC <= '1';
    -- FIN DE LA TRANSMISSION 6
                 ZOUT <= acc_z;

            when 2800 =>
                 READY <= '1';
            when 2802 =>
                 READY <= '0';
```

```
                when others =>
                    null;
            end case;
        end if;
end process;

end Behavioral;
```

```
# SPI signals
# NET "INT1" LOC = "N5";
 NET "SC" LOC = "V4"; #P6
 NET "SDIO" LOC = "T9"; #R3
 NET "SPC" LOC = "V9"; #T3

# onBoard Leds
 NET "LED<0>" LOC = "U18"; # Bank = 1, Pin name = IO_L52N_M1DQ15,
       Sch name = LD0
 NET "LED<1>" LOC = "M14"; # Bank = 1, Pin name = IO_L53P,
                Sch name = LD1
 NET "LED<2>" LOC = "N14"; # Bank = 1, Pin name = IO_L53N_VREF,
       Sch name = LD2
 NET "LED<3>" LOC = "L14"; # Bank = 1, Pin name = IO_L61P,
                Sch name = LD3
 NET "LED<4>" LOC = "M13"; # Bank = 1, Pin name = IO_L61N,
                Sch name = LD4
 NET "LED<5>" LOC = "D4";  # Bank = 0, Pin name = IO_L1P_HSWAPEN_0,
       Sch name = HSWAP/LD5
 NET "LED<6>" LOC = "P16"; # Bank = 1, Pin name = IO_L74N_DOUT_BUSY_1, Sch name = LD6
 NET "LED<7>" LOC = "N12"; # Bank = 2, Pin name = IO_L13P_M1_2,
            Sch name = M1/LD7

 NET "BTN<0>" LOC = "P4";  # Bank = 3, Pin name = IO_L2P,
                Sch name = BTNL : print x on LED
 NET "BTN<1>" LOC = "F5";  # Bank = 3, Pin name = IO_L55N_M3A14,
         Sch name = BTNC : print y on LED
 NET "BTN<2>" LOC = "F6";  # Bank = 3, Pin name = IO_L55P_M3A13,
         Sch name = BTNR : print z on LED

# onBoard SWITCHES
 NET "RESET" LOC = "A10"; # Bank = 0, Pin name = IO_L37N_GCLK12,
    Sch name = SW0

# USB UART Connector
 NET "UART_TXD" LOC = "B16"; # Bank = 0, Pin name = IO_L66P_SCP1, Sch name = USBB-TXD

# clock pin for Atlys rev C board
 NET "CLK"   LOC = "L15"; # Bank = 1, Pin name = IO_L42P_GCLK7_M1UDM, Type = GCLK, Sch
name = GCLK
```

```
#ifdef __cplusplus
    #include <cstdlib>
#else
    #include <stdlib.h>
#endif
#ifdef __APPLE__
#include <SDL/SDL.h>
#else
#include <SDL.h>
#endif

#include <SDL_ttf.h>
#include <windows.h>
#include "define.h"
#include "serial_ctrl.h"
#include "win_serial.h"
#include "videoInput.h"
#include "SDL_rotozoom.h"

int main ( int argc, char** argv )
{
    /* ------------------------------------
    |              init Video            |
    ------------------------------------*/

    //init SDL_tff
    if(TTF_Init() == -1)
    {
        fprintf(stderr, "Erreur d'initialisation de TTF_Init : %s\n", TTF_GetError());
        exit(EXIT_FAILURE);
    }

    // initialize SDL video
    if ( SDL_Init( SDL_INIT_VIDEO | SDL_INIT_TIMER) < 0 )
    {
        printf( "Unable to init SDL: %s\n", SDL_GetError() );
        return 1;
    }

    // make sure SDL cleans up before exit
    atexit(SDL_Quit);

    // create a new window
    const SDL_VideoInfo* screen_info = SDL_GetVideoInfo();
    int screen_h = screen_info->current_h;
    int screen_w = screen_info->current_w;
    SDL_Surface *screen = SDL_SetVideoMode(screen_w, screen_h, 16,
                                SDL_HWSURFACE|SDL_DOUBLEBUF|SDL_FULLSCREEN);

    if ( !screen )
    {
        printf("Unable to set video: %s\n", SDL_GetError());
        return 1;
    }

    /* ------------------------------------
    |           others inits             |
    ------------------------------------*/

    int sensitivity = 8;             //coefficient used for the motion process
    float speed = 0.50;
    bool isCursorPressed = false;   //true when user clic the sensitivity cursor and
start moving it
    bool isCursor2Pressed = false;  //true when user clic the speed cursor and start
```

```
moving it
    bool done = false;              //while done=false, the main loop is running
    bool isLeftHand = false;
    char theStringCpy[6];
    char infoString[40];
    int calibration_x = 0;
    int calibration_y = 0;
    int calibration_z = 0;
    int last_acc_x[4];
    int last_acc_y[4];
    int last_acc_z[4];
    int acc_x;
    int acc_y;
    int acc_z;
    //-----
    if(init_RS232()==1)         //if RS232 can't init then quit immediately : go to
define.h to use prog mode
    {
        #ifndef PROG_MODE
        done = true;
        #endif
    }
    init_timer();

    /* ------------------------------------
    |       init graphic objects         |
    ------------------------------------*/
    bool M_visible = false;
        //------ background image
    SDL_Surface *I_background = NULL;
    SDL_Rect P_background;
    P_background.x = 0;
    P_background.y = 0;
        //------ text : Menu
    TTF_Font *P_palisade = NULL;
    P_palisade = TTF_OpenFont("palisade.ttf", 25);
    SDL_Color couleur_bleu = {51, 153, 255};
    SDL_Surface *texte = TTF_RenderText_Blended(P_palisade, "Menu", couleur_bleu);
    SDL_Rect P_texte;
    P_texte.x = R_P_texte_x;
    P_texte.y = R_P_texte_y;
        //------ headband
    SDL_Surface *I_menu = SDL_LoadBMP("menu.bmp");
    SDL_Rect P_menu;
    P_menu.x = R_P_menu_x;
    P_menu.y = R_P_menu_y;
        //------ bouton quit
    SDL_Surface *I_quit = SDL_LoadBMP("quit.bmp");
    SDL_SetColorKey(I_quit, SDL_SRCCOLORKEY, SDL_MapRGB(I_quit->format, 255, 255, 255));
    SDL_Rect P_quit;
    P_quit.x = R_P_quit_x;
    P_quit.y = R_P_quit_y;
        //------ left hand
    SDL_Surface *I_LHand = SDL_LoadBMP("mainG.bmp");
    SDL_SetColorKey(I_LHand, SDL_SRCCOLORKEY, SDL_MapRGB(I_LHand->format, 255, 0, 0));
    SDL_Rect P_LHand;
    P_LHand.x = R_P_LHand_x;
    P_LHand.y = R_P_LHand_y;
        //------ left hand buttons
    SDL_Surface *I_LHand_off = SDL_LoadBMP("mainG_min_off.bmp");
    SDL_Surface *I_LHand_on = SDL_LoadBMP("mainG_min_on.bmp");
    SDL_Rect P_LHand_min;
    P_LHand_min.x = R_P_LHand_min_x;
    P_LHand_min.y = R_P_LHand_min_y;
```

```
        //------ right hand
    SDL_Surface *I_RHand = SDL_LoadBMP("mainD.bmp");
    SDL_SetColorKey(I_RHand, SDL_SRCCOLORKEY, SDL_MapRGB(I_RHand->format, 255, 0, 0));
    SDL_Rect P_RHand;
    P_RHand.x = R_P_RHand_x;
    P_RHand.y = R_P_RHand_y;
        //------ right hand buttons
    SDL_Surface *I_RHand_off = SDL_LoadBMP("mainD_min_off.bmp");
    SDL_Surface *I_RHand_on = SDL_LoadBMP("mainD_min_on.bmp");
    SDL_Rect P_RHand_min;
    P_RHand_min.x = R_P_RHand_min_x;
    P_RHand_min.y = R_P_RHand_min_y;
        //------ set hand to origine button
    SDL_Surface *I_set0 = SDL_LoadBMP("set_to_0.bmp");
    SDL_SetColorKey(I_set0, SDL_SRCCOLORKEY, SDL_MapRGB(I_set0->format, 0, 0, 0));
    SDL_Rect P_set0;
    P_set0.x = R_P_set0_x;
    P_set0.y = R_P_set0_y;
        //------ blue bar selector for sensitivity
    SDL_Surface *I_selector = SDL_LoadBMP("barre_sensitivity.bmp");
    SDL_SetColorKey(I_selector, SDL_SRCCOLORKEY, SDL_MapRGB(I_selector->format, 255, 255,
255));
    SDL_Rect P_selector;
    P_selector.x = R_P_selector_x;
    P_selector.y = R_P_selector_y;
        //------ red cursor for sensitivity
    SDL_Surface *I_cursor = SDL_LoadBMP("selecteur.bmp");
    SDL_SetColorKey(I_cursor, SDL_SRCCOLORKEY, SDL_MapRGB(I_cursor->format, 255, 255,
255));
    SDL_Rect P_cursor;
    P_cursor.x = R_P_cursor_x;
    P_cursor.y = R_P_cursor_y;
        //------ sensitivity value
    TTF_Font *P_palisade3 = NULL;
    P_palisade3 = TTF_OpenFont("palisade.ttf", 24);
    SDL_Surface *T_sensitivity = TTF_RenderText_Blended(P_palisade3, "8", couleur_bleu);
    SDL_Rect P_T_sensitivity;
    P_T_sensitivity.x = R_P_T_sensitivity_x;
    P_T_sensitivity.y = R_P_T_sensitivity_y;
        //------ blue bar selector for speed
    SDL_Surface *I_selector2 = SDL_LoadBMP("barre_speed.bmp");
    SDL_SetColorKey(I_selector2, SDL_SRCCOLORKEY, SDL_MapRGB(I_selector2->format, 255,
255, 255));
    SDL_Rect P_selector2;
    P_selector2.x = R_P_selector2_x;
    P_selector2.y = R_P_selector2_y;
        //------ red cursor for speed
    SDL_Surface *I_cursor2 = SDL_LoadBMP("selecteur.bmp");
    SDL_SetColorKey(I_cursor2, SDL_SRCCOLORKEY, SDL_MapRGB(I_cursor2->format, 255, 255,
255));
    SDL_Rect P_cursor2;
    P_cursor2.x = R_P_cursor2_x;
    P_cursor2.y = R_P_cursor2_y;
        //------ speed value
    SDL_Surface *T_speed = TTF_RenderText_Blended(P_palisade3, "0.50", couleur_bleu);
    SDL_Rect P_T_speed;
    P_T_speed.x = R_P_T_speed_x;
    P_T_speed.y = R_P_T_speed_y;
        //------ CAM1 buttons
    SDL_Surface *I_CAM1 = NULL;
    SDL_Rect P_CAM1;
    P_CAM1.x = R_P_CAM1_x;
    P_CAM1.y = R_P_CAM1_y;
        //------ CAM2 buttons
```

```
    SDL_Surface *I_CAM2 = NULL;
    SDL_Rect P_CAM2;
    P_CAM2.x = R_P_CAM2_x;
    P_CAM2.y = R_P_CAM2_y;
        //------ black screen button
    SDL_Surface *I_blackScreen = SDL_LoadBMP("black_screen.bmp");
    SDL_SetColorKey(I_blackScreen, SDL_SRCCOLORKEY, SDL_MapRGB(I_blackScreen->format,
255, 255, 0));
    SDL_Rect P_blackScreen;
    P_blackScreen.x = R_P_blackScreen_x;
    P_blackScreen.y = R_P_blackScreen_y;
            //------ text : x y z
    TTF_Font *P_palisade2 = NULL;
    P_palisade2 = TTF_OpenFont("palisade.ttf", 30);
    SDL_Color couleur_vert = {0, 100, 0};
    SDL_Surface *T_xyz = TTF_RenderText_Blended(P_palisade2, "x", couleur_vert);
    SDL_Rect P_T_xyz;
    P_T_xyz.x = R_P_T_xyz_x;
    P_T_xyz.y = R_P_T_xyz_y;

    /* ----------------------------------
    |          init VideoInput          |
    ----------------------------------*/
     videoInput VI;
     switch (VI.listDevices())
     {
         case 0:
         I_CAM1 = SDL_LoadBMP("CAM1_off.bmp");
         I_CAM2 = SDL_LoadBMP("CAM2_off.bmp");
         break;

         case 1:
         I_CAM1 = SDL_LoadBMP("CAM1_on.bmp");
         I_CAM2 = SDL_LoadBMP("CAM2_off.bmp");
         break;

         default:
         I_CAM1 = SDL_LoadBMP("CAM1_on.bmp");
         I_CAM2 = SDL_LoadBMP("CAM2_on.bmp");
     }
     SDL_SetColorKey(I_CAM1, SDL_SRCCOLORKEY, SDL_MapRGB(I_CAM1->format, 255, 255, 0));
     SDL_SetColorKey(I_CAM2, SDL_SRCCOLORKEY, SDL_MapRGB(I_CAM2->format, 255, 255, 0));

    /* ----------------------------------
    |         program main loop         |
    ----------------------------------*/
    while (!done)
    {
        // message processing loop
        SDL_Event event;
        while (SDL_PollEvent(&event))
        {
            // check for messages
            switch (event.type)
            {
                // exit if the window is closed
            case SDL_QUIT:
                done = true;
                break;

                // check for keypresses
            case SDL_KEYDOWN:
                {
                    // exit if ESCAPE is pressed
```

```
                    if (event.key.keysym.sym == SDLK_ESCAPE)
                        done = true;
                    break;
                }
            case SDL_MOUSEMOTION:
                {
                    if(event.motion.y > 120 && (isCursorPressed==false))
                    {
                        M_visible = false;
                    }
                    else
                    {
                        if(event.motion.y==0)
                        {
                            M_visible = true;
                        }
                    }
                    if(isCursorPressed)
                    {
                        if(event.motion.x<R_P_selector_x+26)
                        {
                            P_cursor.x = R_P_selector_x+10;
                        }
                        else
                        {
                            if(event.motion.x>R_P_selector_x+477)
                            {
                                P_cursor.x = R_P_selector_x+460;
                            }
                            else
                            {
                                P_cursor.x = event.motion.x-16;
                                sensitivity = (int)((event.motion.x - (R_P_selector_x +
26))/23);   //we make the sensitivity value between 0 and 20
                            }
                        }
                        char value[3];
                        sprintf(value, "%d", sensitivity);
                        T_sensitivity = TTF_RenderText_Blended(P_palisade3, value,
couleur_bleu);
                    }

                    if(isCursor2Pressed)
                    {
                        if(event.motion.x<R_P_selector2_x+26)
                        {
                            P_cursor2.x = R_P_selector2_x+10;
                        }
                        else
                        {
                            if(event.motion.x>R_P_selector2_x+477)
                            {
                                P_cursor2.x = R_P_selector2_x+460;
                            }
                            else
                            {
                                P_cursor2.x = event.motion.x-16;
                                speed = (event.motion.x - (R_P_selector2_x + 26.0))/
150.3;   //we make the speed value between 0 and 3
                            }
                        }
                        char value[5];
                        sprintf(value, "%.2f", speed);
                        T_speed = TTF_RenderText_Blended(P_palisade3, value,
```

```
couleur_bleu);
                    }

                    break;
                }
            case SDL_MOUSEBUTTONUP:
                {
                    if(event.button.x>R_P_quit_x && event.button.x<R_P_quit_x+77 &&
event.button.y>R_P_quit_y &&    //quit
                       event.button.y<R_P_quit_y+44 && M_visible)
                    {
                       done = true;
                    }
                    else
                    {
                        if(event.button.x>R_P_LHand_min_x &&
event.button.x<R_P_LHand_min_x+140 && event.button.y>R_P_LHand_min_y &&  //activate left
hand
                            event.button.y<R_P_LHand_min_y+80 && M_visible)
                        {
                            isLeftHand = true;
                        }
                        else
                        {
                            if(event.button.x>R_P_RHand_min_x &&
event.button.x<R_P_RHand_min_x+140 && event.button.y>R_P_RHand_min_y &&  //activate right
hand
                            event.button.y<R_P_RHand_min_y+80 && M_visible)
                            {
                                isLeftHand = false;
                            }
                            else
                            {
                                if(event.button.x>R_P_set0_x && event.button.x<R_P_set0_x
+99 &&      //move hand to origine
                                   event.button.y>R_P_set0_y && event.button.y<R_P_set0_y
+89 && M_visible)
                                {
                                    P_LHand.x = R_P_LHand_x;
                                    P_LHand.y = R_P_LHand_y;
                                    P_RHand.x = R_P_RHand_x;
                                    P_RHand.y = R_P_RHand_y;
                                    calibration_x = theStringCpy[2];
                                    calibration_y = theStringCpy[3];
                                    calibration_z = theStringCpy[4];
                                }
                                else
                                {
                                    if(event.button.x>R_P_CAM1_x &&
event.button.x<R_P_CAM1_x+68 && event.button.y>R_P_CAM1_y && //take picture from CAM1
                                        event.button.y<R_P_CAM1_y+45 && VI.listDevices()
>0 && M_visible)
                                    {
                                        double coef_rotozoom;  //this is the coefficient
to use in the zoomSurface() function to make the picture full screen
                                        VI.setupDevice(0);
                                        VI.setupDevice(0, 640, 480);
                                        int cam0_width = VI.getWidth(0);
                                        int cam0_height = VI.getHeight(0);
                                        int cam0_size = VI.getSize(0);
                                        if ((screen_w/cam0_width) <
(screen_h<cam0_height))     //find coef_rotozoom and place the picture in the center of
the screen
                                        {
```

```
                                        coef_rotozoom = (double)screen_w/(double)
cam0_width;
                                        P_background.y = (screen_h - (int)
(coef_rotozoom*cam0_height))/2;
                                    }
                                    else
                                    {
                                        coef_rotozoom = (double)screen_h/(double)
cam0_height;
                                        P_background.x = (screen_w -
(coef_rotozoom*cam0_width))/2;
                                    }
                                    unsigned char* cam0_buffer = new unsigned char
[cam0_size];
                                    while(!VI.isFrameNew(0));
                                        VI.getPixels(0, cam0_buffer, false, true);
                                    I_background = SDL_CreateRGBSurfaceFrom
(cam0_buffer, cam0_width, cam0_height, (cam0_size/(cam0_height*cam0_width))*8, cam0_size/
cam0_height, 0, 0, 0, 0);
                                    I_background = zoomSurface(I_background,
coef_rotozoom, coef_rotozoom, 1);
                                    VI.stopDevice(0);
                                }
                                else
                                {
                                    if(event.button.x>R_P_CAM1_x &&
event.button.x<R_P_CAM1_x+68 &&  event.button.y>R_P_CAM1_y+46 &&//acces settings of CAM1
                                        event.button.y<R_P_CAM1_y+92 &&
VI.listDevices()>0 && M_visible)
                                    {
                                        VI.setupDevice(0);
                                        VI.showSettingsWindow(0);
                                        VI.stopDevice(0);
                                    }
                                    else
                                    {
                                        if(event.button.x>R_P_CAM2_x &&
event.button.x<R_P_CAM2_x+68 &&  event.button.y>R_P_CAM2_y &&//take picture from CAM2
                                            event.button.y<R_P_CAM2_y+45 &&
VI.listDevices()>1 && M_visible)
                                        {
                                            double coef_rotozoom;  //this is the
coefficient to use in the zoomSurface() function to make the picture full screen
                                            VI.setupDevice(1);
                                            VI.setupDevice(1, 640, 480);
                                            int cam1_width = VI.getWidth(1);
                                            int cam1_height = VI.getHeight(1);
                                            int cam1_size = VI.getSize(1);
                                            if ((screen_w/cam1_width) <
(screen_h<cam1_height))     //find coef_rotozoom and place the picture in the center of
the screen
                                            {
                                                coef_rotozoom = (double)screen_w/
(double)cam1_width;
                                                P_background.y = (screen_h - (int)
(coef_rotozoom*cam1_height))/2;
                                            }
                                            else
                                            {
                                                coef_rotozoom = (double)screen_h/
(double)cam1_height;
                                                P_background.x = (screen_w -
(coef_rotozoom*cam1_width))/2;
                                            }
```

```
                                                unsigned char* cam1_buffer = new unsigned
char[cam1_size];
                                                while(!VI.isFrameNew(1));
                                                    VI.getPixels(1, cam1_buffer, false,
true);
                                                I_background = SDL_CreateRGBSurfaceFrom
(cam1_buffer, cam1_width, cam1_height, (cam1_size/(cam1_height*cam1_width))*8, cam1_size/
cam1_height, 0, 0, 0, 0);
                                                I_background = zoomSurface(I_background,
coef_rotozoom, coef_rotozoom, 1);
                                                VI.stopDevice(1);
                                            }
                                            else
                                            {
                                                if(event.button.x>R_P_CAM2_x &&
event.button.x<R_P_CAM2_x+68 &&  event.button.y>R_P_CAM2_y+46 &&    //acces settings of
CAM2
                                                    event.button.y<R_P_CAM2_y+92 &&
VI.listDevices()>1 && M_visible)
                                                {
                                                    VI.setupDevice(1);
                                                    VI.showSettingsWindow(1);
                                                    VI.stopDevice(1);
                                                }
                                                else
                                                {
                                                    if(event.button.x>R_P_blackScreen_x
&& event.button.x<R_P_blackScreen_x+84 &&  event.button.y>R_P_blackScreen_y &&    //black
screen
                                                        event.button.y<R_P_blackScreen_y
+58 && M_visible)
                                                    {
                                                        unsigned char* blackscreen = new
unsigned char[screen_h*screen_w];
                                                        for(int z=0; z<screen_h*screen_w;
z++)
                                                        {
                                                            blackscreen[z]=0xFF;
                                                        }
                                                        I_background =
SDL_CreateRGBSurfaceFrom(blackscreen, screen_w, screen_h, 1, screen_w, 0, 0, 0, 0);
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                        if(isCursorPressed)         //sensitivity
                        {
                            isCursorPressed = false;
                        }
                        if(isCursor2Pressed)        //speed
                        {
                            isCursor2Pressed = false;
                        }
                        break;
                    }
                case SDL_MOUSEBUTTONDOWN:
                    {
                        if(event.button.x>P_cursor.x && event.button.x<P_cursor.x+33 &&
event.button.y>P_cursor.y && event.button.y<P_cursor.y+39 && M_visible)
```

```
                    {
                        isCursorPressed = true;
                    }
                    if(event.button.x>P_cursor2.x && event.button.x<P_cursor2.x+33 &&
event.button.y>P_cursor2.y && event.button.y<P_cursor2.y+39 && M_visible)
                    {
                        isCursor2Pressed = true;
                    }
                }
            case SDL_USEREVENT:
                {
                    //future algorithms sould make a different process depending on the
selected hand
                //    if(isLeftHand)
                //    {
                      strcpy(theStringCpy, get_theString());
                      acc_x = theStringCpy[2];
                      acc_y = theStringCpy[3];
                      acc_z = theStringCpy[4];

                      //acc_y = acc_y - calibration_y;
                      if (abs(acc_y - calibration_y) > sensitivity)
                          acc_y = acc_y - calibration_y;
                      else
                          acc_y = 0;

                      if(P_LHand.x <= 0)
                      {
                          if (((last_acc_y[3]+2*last_acc_y[1]+acc_y)/4) > 0)
                          {
                              P_LHand.x = P_LHand.x + (last_acc_y[3]+2*last_acc_y
[1]+acc_y)*speed;
                          }
                      }
                      else
                      {
                          if (P_LHand.x>=940)
                          {
                              if (((last_acc_y[3]+2*last_acc_y[1]+acc_y)/4) < 0)
                              {
                                  P_LHand.x = P_LHand.x + (last_acc_y[3]+2*last_acc_y
[1]+acc_y)*speed;
                              }
                          }
                          else
                          {
                              P_LHand.x = P_LHand.x + (last_acc_y[3]+2*last_acc_y
[1]+acc_y)*speed;
                          }
                      }

                      last_acc_y[3] = last_acc_y[2];
                      last_acc_y[2] = last_acc_y[1];
                      last_acc_y[1] = last_acc_y[0];
                      last_acc_y[0] = acc_y;
                //    }
                //    else
                //    {

                //    }
                }

            } // end switch
        } // end of message processing
```

```
        // ---------- CLEAR SCREEN --------- //
        SDL_FillRect(screen, 0, SDL_MapRGB(screen->format, 0, 0, 0));

        // ---- AND DRAW A FRESH SCREEN ---- //
        SDL_BlitSurface(I_background, NULL, screen, &P_background);
        if(isLeftHand)
        {
            SDL_BlitSurface(I_LHand, NULL, screen, &P_LHand);
        }
        else
        {
            SDL_BlitSurface(I_RHand, NULL, screen, &P_RHand);
        }
        if(M_visible)
        {
            SDL_BlitSurface(I_menu, NULL, screen, &P_menu);
            SDL_BlitSurface(texte, NULL, screen, &P_texte); /* Blit du texte */
            SDL_BlitSurface(I_quit, NULL, screen, &P_quit);
            SDL_BlitSurface(I_set0, NULL, screen, &P_set0);
            SDL_BlitSurface(I_selector, NULL, screen, &P_selector);
            SDL_BlitSurface(I_cursor, NULL, screen, &P_cursor);
            SDL_BlitSurface(T_sensitivity, NULL, screen, &P_T_sensitivity);
            SDL_BlitSurface(I_selector2, NULL, screen, &P_selector2);
            SDL_BlitSurface(I_cursor2, NULL, screen, &P_cursor2);
            SDL_BlitSurface(T_speed, NULL, screen, &P_T_speed);
            SDL_BlitSurface(I_CAM1, NULL, screen, &P_CAM1);
            SDL_BlitSurface(I_CAM2, NULL, screen, &P_CAM2);
            SDL_BlitSurface(I_blackScreen, NULL, screen, &P_blackScreen);
            if(isLeftHand)
            {
                SDL_BlitSurface(I_LHand_on, NULL, screen, &P_LHand_min);
                SDL_BlitSurface(I_RHand_off, NULL, screen, &P_RHand_min);
            }
            else
            {
                SDL_BlitSurface(I_LHand_off, NULL, screen, &P_LHand_min);
                SDL_BlitSurface(I_RHand_on, NULL, screen, &P_RHand_min);
            }
        }
        else
        {
            sprintf(infoString, " x:%d     y:%d     z:%d     %s", acc_x, acc_y, acc_z,
theStringCpy);
            T_xyz = TTF_RenderText_Blended(P_palisade2, infoString, couleur_vert);
            SDL_BlitSurface(T_xyz, NULL, screen, &P_T_xyz);
        }

        // DRAWING ENDS HERE

        // finally, update the screen :)
        SDL_Flip(screen);
    } // --- END MAIN LOOP ---

    //everyone is now free !
    SDL_FreeSurface(I_menu);   /* On libère la surface */
    SDL_FreeSurface(I_quit);
    SDL_FreeSurface(I_LHand);
    SDL_FreeSurface(I_LHand_off);
    SDL_FreeSurface(I_LHand_on);
    SDL_FreeSurface(I_RHand);
    SDL_FreeSurface(I_RHand_off);
    SDL_FreeSurface(I_RHand_on);
```

```
    SDL_FreeSurface(I_set0);
    TTF_CloseFont(P_palisade);
    TTF_Quit();
    stop_timer();
    stop_RS232();

    // all is well ;)
    printf("Exited cleanly\n");
    return 0;
}
```

```
/* this line allow you to use the application           */
/* without the FPGA plugged. The behaviour could       */
/* be affected but you won't be stuck in front of the  */
/* "Initialization. Plug and switch on the FPGA" screen */
/* Just uncomment this line and clic the "STOP" button  */
#define PROG_MODE

//Position of the value x y z
#define R_P_T_xyz_x 0
#define R_P_T_xyz_y 0

    /* Position of the elements of the  headband from left to right */
//Position of headband
#define R_P_menu_x 0
#define R_P_menu_y 0
//Position of texte Menu
#define R_P_texte_x 7
#define R_P_texte_y 7
//Position of button 'quit'
#define R_P_quit_x 7
#define R_P_quit_y 42
//Position of button 'activate left hand'
#define R_P_LHand_min_x 150
#define R_P_LHand_min_y 10
//Position of button 'activate right hand'
#define R_P_RHand_min_x 300
#define R_P_RHand_min_y 10
//set position of the hand to origine
#define R_P_set0_x 450
#define R_P_set0_y 10
//blue selector for sensitivity (1)
#define R_P_selector_x 560
#define R_P_selector_y 10
//red cursor for sensitivity (1)
#define R_P_cursor_x 750
#define R_P_cursor_y 10
//sensitivity value
#define R_P_T_sensitivity_x 1159
#define R_P_T_sensitivity_y 14
//blue selector for speed (2)
#define R_P_selector2_x 560
#define R_P_selector2_y 55
//red cursor for speed (2)
#define R_P_cursor2_x 645
#define R_P_cursor2_y 55
//speed value
#define R_P_T_speed_x 1120
#define R_P_T_speed_y 59
//CAM1 buttons
#define R_P_CAM1_x 1190
#define R_P_CAM1_y 5
//CAM2 buttons
#define R_P_CAM2_x 1265
#define R_P_CAM2_y 5
//black screen button
#define R_P_blackScreen_x 1340
#define R_P_blackScreen_y 20

    /* Initial position of the hand */
//Left hand
#define R_P_LHand_x 400
#define R_P_LHand_y 200
//Right hand
#define R_P_RHand_x 400
```

```
#define R_P_RHand_y 200
```

```
#include "serial_ctrl.h"

SDL_TimerID timer;
char param_bidon = 0;    //useless parameter to try the SDL_AddTimer fonction
HANDLE fd;               //Hadle to open RS232
char theResult[7];
char theString[7];
int result;              //if != 0 then data on RS232 OK

SDL_Thread *rThread = NULL;
SDL_mutex *theResultLock = NULL;

/* ------------------------------------
|               RS232                |
------------------------------------*/
int init_RS232(void)
{
    //------ bouton STOP
    SDL_Surface *I_STOP = SDL_LoadBMP("STOP.bmp");
    SDL_Rect P_STOP;
    P_STOP.x = 800;
    P_STOP.y = 0;
    //------ text : init
    TTF_Font *P_palisade_init = NULL;
    P_palisade_init = TTF_OpenFont("palisade.ttf", 40);
    SDL_Color couleur_rouge = {255, 0, 0};
    SDL_Surface *texte_init = TTF_RenderText_Blended(P_palisade_init, "Initialization.
Plug and switch on FPGA", couleur_rouge);
    SDL_Rect P_texte_init;
    P_texte_init.x = 10;
    P_texte_init.y = 30;
    SDL_BlitSurface(texte_init, NULL, SDL_GetVideoSurface(), &P_texte_init);
    SDL_BlitSurface(I_STOP, NULL, SDL_GetVideoSurface(), &P_STOP);
    SDL_Flip(SDL_GetVideoSurface());
    //search for the right COM port
    char string[15];
    int i;
    Sleep(600);
    fd = NULL;
    while(fd == NULL)
    {
        for(i=0; i<15; i++)
        {
            sprintf(string, "\\\\.\\COM%d", i);
            fd = OpenCom(string, 9600, 8, 1);
            if(fd != NULL)
            {
                theResultLock = SDL_CreateMutex();
                char *tnames[1] = {"Thread1"};
                rThread = SDL_CreateThread(constantRead, tnames[0]);
                break;
            }
            SDL_Event event;
            while (SDL_PollEvent(&event))
            {
                if(event.type == SDL_MOUSEBUTTONUP)
                {
                    if(event.button.x>800 && event.button.x<996 && event.button.y>0 &&
event.button.y<105)
                    {
                        return 1;
                    }
                }
            }
```

```
        }
    }
    return 0;
}

int constantRead(void* data)
{
    char buffer[6];
    while(1)
    {
        result = ReceiveCom(fd, buffer, 5);
        SDL_LockMutex(theResultLock);
        strcpy(theResult, buffer);
        SDL_UnlockMutex(theResultLock);
    }
    return 0;
}

void stop_RS232(void)
{
    CloseCom(fd);
    SDL_DestroyMutex(theResultLock);
}


/* ------------------------------------
|                Timer                |
------------------------------------*/
//init the timer that get data from RS232
void init_timer(void)
{
    //get data every 20ms (make this value similar in VHDL)
    timer = SDL_AddTimer(20, get_RS232, &param_bidon);
}

Uint32 get_RS232(Uint32 interval, void *param)
{
    char buffer[6];
    int i;
    SDL_LockMutex(theResultLock);
    strcpy(buffer, theResult);
    SDL_UnlockMutex(theResultLock);

    for(i=0; i<5; i++)
    {
        if(buffer[i]=='A' && buffer[(i+1)%5]=='d')
        {
            theString[0]=buffer[i];
            theString[1]=buffer[(i+1)%5];
            theString[2]=buffer[(i+2)%5];
            theString[3]=buffer[(i+3)%5];
            theString[4]=buffer[(i+4)%5];
            theString[5]='\0';
            break;
        }
    }



    SDL_Event event;
    SDL_UserEvent userevent;

    // We push a new event SDL_USEREVENT in the stack
    userevent.type = SDL_USEREVENT;
```

```
    userevent.code = 0;
    userevent.data1 = NULL;
    userevent.data2 = NULL;

    event.type = SDL_USEREVENT;
    event.user = userevent;

    SDL_PushEvent(&event);

    // Please call me back in interval
    return(interval);
}

void stop_timer(void)
{
    SDL_RemoveTimer(timer);
}

/* ------------------------------------
|         Get accelerations          |
------------------------------------*/

char* get_theString(void)
{
    return theString;
}
```

```
#ifndef  H_PROCESS_MOVE
#define H_PROCESS_MOVE

#ifdef __APPLE__
#include <SDL/SDL.h>
#else
#include <SDL.h>
#endif

#ifdef __cplusplus
    #include <cstdlib>
#else
    #include <stdlib.h>
#endif
#ifdef __APPLE__
#include <SDL/SDL.h>
#else
#include <SDL.h>
#endif

#include <SDL_ttf.h>
#include <SDL_thread.h>
#include <SDL_mutex.h>
#include <windows.h>
#include "win_serial.h"

/* ------------------------------------
|               RS232                |
------------------------------------*/
//init and find the right COM port
int init_RS232(void);

//close RS232
void stop_RS232(void);

//thread
int constantRead(void* data);

/* ------------------------------------
|                Timer               |
------------------------------------*/

//init the timer that get data from RS232
void init_timer(void);

//callback fonction
Uint32 get_RS232(Uint32 intervalle, void *parametre);

//stop timer
void stop_timer(void);


/* ------------------------------------
|         Get accelerations          |
------------------------------------*/

char* get_theString(void);

#endif
```

```
#include "win_serial.h"

HANDLE OpenCom(char *port,int speed,int size,int stopBits)
{
  DCB g_DCB;
  HANDLE fd;

  fd = CreateFile(port, GENERIC_READ | GENERIC_WRITE,
0,NULL,OPEN_EXISTING,FILE_FLAG_WRITE_THROUGH | FILE_FLAG_NO_BUFFERING,NULL);

  if(fd == INVALID_HANDLE_VALUE)
  {
    return NULL;
  }
  else
  {
    PurgeComm(fd, PURGE_TXABORT|PURGE_RXABORT|PURGE_TXCLEAR|PURGE_RXCLEAR);

    g_DCB.DCBlength = sizeof(DCB);

    GetCommState(fd, &g_DCB);

    g_DCB.BaudRate = speed;
    g_DCB.ByteSize = size;

    g_DCB.fDtrControl = DTR_CONTROL_DISABLE; // (DISABLE)

    /*if (strcmp(Parite,"Aucune")==0)
    g_DCB.Parity = NOPARITY;
    if (strcmp(Parite,"Paire")==0)
    g_DCB.Parity = EVENPARITY;
    if (strcmp(Parite,"Impaire")==0)
    g_DCB.Parity = ODDPARITY;*/
    g_DCB.Parity = NOPARITY;

    // Gestion du Stop Bit
    if ( stopBits == 1 )
    g_DCB.StopBits = ONESTOPBIT;
    else if ( stopBits == 2 )
    g_DCB.StopBits = TWOSTOPBITS;

    SetCommState(fd, &g_DCB);

    return fd;
  }
}

int SendCom(HANDLE fd, char* data)
{
  DWORD NumBytes = 0;

  if(fd != NULL)
    if (WriteFile(fd, data, strlen(data),&NumBytes,NULL) != 0)
      return NumBytes;

  return 0;
}


int ReceiveCom(HANDLE fd, char *data, int size)
{
     COMSTAT Stat;
     DWORD Errors;
     int avBytes = 0;
```

```
	long unsigned int readBytes = 0;

	if(fd != NULL)
	{
		//Pour éviter de gérer un time out
		Sleep(1);
		//Pour connaitre le nombre d'octets dans le buffer d'entrée
		ClearCommError(fd,&Errors,&Stat);
		avBytes = Stat.cbInQue;

		//On effectue la lecture si il y a des caractères présents
		if(avBytes > size)
		{
			//Reception de la Chaine
			if(ReadFile(fd, data, size, &readBytes, NULL) == 0)
				return 0;
		}
		//Finition de la Chaine
		data[readBytes] = '\0' ;
		return readBytes;
	}
	else
		return 0;

}

int CloseCom(HANDLE fd)
{
  if(fd != NULL)
  {
    CloseHandle(fd);
    return 1;
  }
  return 0;
}
```

```
#include <windows.h>
#include <winbase.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

HANDLE OpenCom(char *port,int speed,int size,int stopBits);
int SendCom(HANDLE fd, char* data);
int ReceiveCom(HANDLE fd, char *data, int size);
int CloseCom(HANDLE fd);

/* Exemple
 *
 * HANDLE fd;
 * char buffer[1025];
 * int result;
 *
 * fd = OpenCom("\\\\.\\COM1", 9600, 8, 1);
 * result = SendCom(fd, buffer); //Return nb of bytes sent
 * result = ReceiveCom(fd, buffer, 1024); //Return nb of bytes read
 * CloseCom(fd);
 *
 */
```