University Carlos III of Madrid
Higher Polytechnic School

Industrial Electronics and Automation Degree

Final Degree Project

CONTROL OF A 6DOF ROBOTIC ARM USING LABVIEW

Author: Cristina Muñoz Hernández
Tutor: Luis Enrique Moreno Llorente

Leganés, September 2015

Title: Control of a 6DOF Robotic Arm using LabVIEW

Theme: Final Degree Project

Author: Cristina Muñoz Hernández

Tutor: Luis Enrique Moreno Llorente

University Carlos III, Madrid

Campus Leganés

FINAL DEGREE PROJECT

Engineering Department in Systems and Automation

University Carlos III of Madrid

Title: Control of a 6DOF Robotic Arm Using LabVIEW

Author: Cristina Muñoz Hernández

Tutor: Luis Enrique Moreno Llorente

The defense of this Final Degree Project is dated on the date:

_____

The magistrates' court was composed by:

- President: _____
- Secretary: _____
- Spokesperson: _____

Having obtained the rating: _____

President **Secretary** **Spokesperson**

Abstract

Real Time means in time. In other words, a real-time system ensures that responses occur in time, or on time. With general purpose operating systems, you cannot ensure that a response occurs within any given time period, and calculations might finish much later or earlier than expected

Nowadays there are many applications that need to be executed in time. In this project, the idea is to make a first approach of a Real Time system, the environments that allowed this kind of control and how a complete project is created; step by step.

It has been printed a 3D robotic arm that will be used to strengthen the concepts explained by designing a complete applications to control it. In order to control this prototype it will be used the myRIO, a device oriented to be used by students that are interested in control applications developed in LabVIEW.

In this report it is explained from the theoretical concepts to the practical ones developed in LabVIEW, going through the use of different available tools used for transferring data from one part of the controller to the other.

General Index

Abbreviation Index

AI	**A**nalog **I**nput
AO	**A**nalog **O**utput
API	**A**pplication **P**rogramming **I**nterface
ARCP	**A**dvanced **R**apid **C**ontrol **P**rototyping
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
CASE	**C**omputer **A**ided **S**oftware **E**ngineering
CAN	**C**ontroller **A**rea **N**etwork
cRIO	**C**ompact **R**econfigurable **I**nput **O**utput
CPU	**C**entral **P**rocessing **U**nit
DMA	**D**irect **M**emory **A**ccess
DI	**D**igital **I**nput
DO	**D**igital **O**utput
DOF	**D**egree **O**f **F**reedom or **D**epartment **O**f **D**efense
ECM	**E**mbedded **C**ontrol and **M**anagement
ECU	**E**ngine **C**ontrol **U**nit
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**escription **L**anguage
HMI	**H**uman **M**achine **I**nterface
I/O	**I**nput-**O**utput
IRQ	**I**nterrupt **Req**uest
LabVIEW	**L**aboratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench
LIN	**L**ocal **I**nterconnect **N**etwork
LVRT	**L**ab**V**IEW **R**eal **T**ime
MATLAB	**MAT**rix **LAB**oratory
PCI	**P**eripheral **C**omponent **I**nterconnect
PID	**P**roporcional-**I**ntegral-**D**erivativo
PIL	**P**rocessor **I**n the **L**oop
RAM	**R**andom **A**ccess**M**emory
RCP	**R**apid **C**ontrol **P**rototyping
RIO	**R**econfigurable **I**nput **O**utput
ROM	**R**ead **O**nly **M**emory
RT	**R**eal **T**ime
RTOS	**Real**-**T**ime**O**perating **S**ystem
SPI	**S**erial **P**eripheral **I**nterface
UART	**U**niversal **A**syncronous **R**eceiver/**T**ransmitter
UML	**U**nified**M**odeling **L**anguage
USB	**U**niversal **S**erial **B**us
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage
VHSIC	**V**ery **H**igh Speed **I**ntegrated **C**ircuits
VI	**V**irtual **I**nstrument
XCP	E**x**tended **C**alibration **P**rotocol

Figure Index

Tables & Equation Indexes

Table Index

Equation Index

Chapter 1: Preamble & Goals

Chapter 1: Preamble and Goals

1.1 Introduction

On the trade of continuous evolution in industrial technology, day by day the importance of developing new techniques increases. Since 1930s some human-developed task are done by human-made machines. These tasks used to be the most dangerous or difficult ones through the manufacturing line.

Nowadays, the importance of technology in many industrial applications has increase in the way that there are some that cannot be done without some devices. This is the case of acquiring data at very high velocity or manipulation toxic components.

Now, let's think in the case of industrial robots; more precisely, in robotic arms. What would happen if for some reason they are not previously tested? Which consequences may have if one of these devices does not stop when a user press the emergency bottom? The objective of these two questions is to introduce the reader to the very beginning of this project.

On one hand, during these months the idea was to improve an already design prototype of a 6dof printable arm. For doing this, the main change is to replace the controller and introduce a powerful type of control: Real Time control.

On the other hand, the idea was to learn how to implement a complete application with a control different from what is learn during the degree. In this approach, the environment used is LabVIEW, one of the most-used tools in industry, and the controller myRIO. This controller is a device that allows students to learn and be familiarized with this environment but is not design to use it for bigger applications.

1.2 Goals

The objective of this section is to fix the main objectives searched in this project. Firstly, as it has been said, the aim of this project is to introduce the concept of Real-Time controllers and its different possibilities.

For this, first it will be presented the Real Time concept and some of the possibilities that are nowadays offered in the market. What is more, the reader will go around the most important features of the Real Time environments, and afterwards focus in LabVIEW and its different Modules.

In order to support all these concepts a robotic arm printed with a 3D printer in the university will be controlled.

The idea behind this project is to find equilibrium of research and design of code. In this Final Degree Project the objective is not only describe and initiate some concepts of Control, but also to put in practice the concept that had been illustrated by controlling real hardware.

One of the things that must be considered in this project is that the objective is to illustrate the main concepts of Real Time control. Despite the fact that it will be used a robotic arm, this hardware does not need such a powerful control. As it will be mentioned, this kind of device may be controlled by an Arduino and using other kind of control.

The documentation used comes from two main flows. On one hand, for supporting theoretical concepts the main research is made from other UC3M thesis from the same Department and also from the National Instruments Online Documentation. On the other hand, the main webpage to learn how to program are LabVIEW Manuals and webpage of the company that created the controller (National Instruments).

1.3 Developing phases

In this section it is explained the different parts in which the process may be splitted. First, graphically I represent the approximate time that I have spend in each single task over a total time of project development, taking into account that the work started in February and ends up in September.

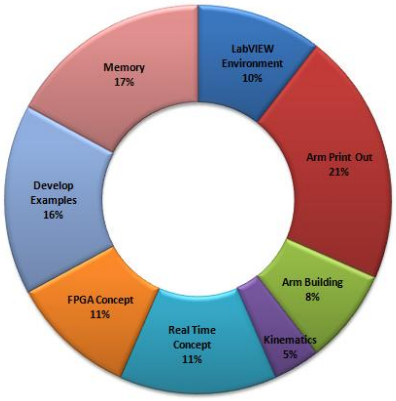

Figure 1. Developing Phases

In its correspondent order the several phases that composed this project are:

1. Real Time concept: this part is the main one and for that reason the first that was needed. As it has been said the objective of this to understand and have a general view of what Real Time is.

   However, this part may be divided in two different phases: the one that includes a study of what is Real Time Concept and how it can be applied to control, and the other one is the one that concerns the knowledge of the Real Time Module used: the one that is included in LabVIEW environment. For this part there will be used some thesis from the UC3M such as the one from Antonio Flores and also the manuals of the LabVIEW Real Time Module.

2. LabVIEW: This is the environment that will be used for developing the examples. The time spent in this phase was basically used for getting used to the software and its graphical programming.

3. Arm Print Out: as it can be seen in the graph, this is the phase that takes the largest amount of time. This will be explained along other chapters but it can be explained by saying that the print put process was in the university and the printers used were not always available.

4. Arm Building: This process concerns to the building part. It consist in join all the pieces together and include the seven servo motors in order to have the complete arm assembled and connections and prepared to use.

5. Kinematics: in order to be able to control the robotic arm it is needed to know the theoretical basis on how does it moves and which limitation does it have. For his section the Final Degree Project made by Antonio Castro and the kinematic study of this arm made by Carlos Rodriguez.

6. FPGA: this is one of the most challenge parts for this project. In this phase it starts from zero studying the concept of FPGA and how do they usually work and then, I have made the online FPGA training available for students from National Instruments.

   Finally, this section includes some previous simple projects developed in order to being familiarized with this tool before trying to control the arm.

7. Developing Examples: in this part is included the two practical examples designed in order to understand the differences between traditional control and Real Time one.

   Additionally, in these two examples are shown some differences between both platforms. The main one is the difference between having a code only run in hardware or a system which has some hardware support but needs also to have some special software available.

8. Memory: after having all the ideas, the concepts obtained in the training and the examples implemented, it is the time to write up so that anyone that is interested in this topic may have all these information putted together.

1.4 Tools

The objective of this section is mainly highlighting the tools that are used in the whole project and whether they would be explained later in any chapter or not, have a brief explanation about them.

In this section there will be mentioned these tools that have done this project being able. In order to achieve the objectives that have already mentioned, along the complete development there have been used the following tools:

1.4.1 Thingverse

Thingverse is a web page that is used by users of 3D printers, laser cutters, milling machines, not only people that design each own pieces, but also by people like me in this project that is looking for a specific object.

It is a website which objective is sharing digital design files created by many users. It was created in 2008 and nowadays has more than 400 million designs uploaded. It provides open source hardware designs and the idea is not only to allow other users to build the design but also to invite others to improve the design.

http://www.thingiverse.com/

The idea of this platform is creating a place where you can search by different categories and download some of the designs in order to printed it or maybe make some customization first. This is a tool that is specially mentioned because has been made possible to show the control of a robotic arm without design it.

The advantage of this website is that you can choose from more than one design in order to have the design that fix better in the application you want to develop. In case of robotic arm there were three that may have been chosen regarding to the features that were looked for.

This tool was also used to compare different functionalities for robotic arms that were already design. However, none of them fulfill the expectations and for this reason, the last link of the arm was personally design to obtain a simple functionality that allows the reader to see how Real Time Control Works.

1.4.2 Printing tools

For this project the robotic arm was printed in the university. For this reason, there have been used three different tools for the printing process. In the beginning, the idea was to print every piece using the same printer, but in order to save time and avoid breakdowns; the final decision was to use both available printers.

The idea of this section is to describe the main features of both printers and the tools used in order to be able to print. This will be important because, three printers mean different tools to control them.

The objective is to describe the differences and common characteristics of both tools. Although the printers that have been used are explained in the Printing Process Section from chapter tools, it is necessary to know which software allows print with them the pieces downloaded.

Before explaining the tools used it is important to clarify how them work. The procedure for printing is similar for every 3D printer; one of the common things is that the piece design may be stored in different file formats: GCode and stl are the two of them that are used in this project.

In both cases, the piece at the beginning will be in a .stl format. STL stand for Stereo Lithography, this kind of file can be generated in any CAD software and it defines the geometry of the piece that will be printed. This format does not included color, texture and physical properties information.

In both tools, the stl will be loaded from the program. These environments allow the user to visualize the piece that will be printed and to resize it if needed. Afterwards, there will be generated a GCode.

GCode files are files with which user tell computerized machine tools how to make the piece. This file contains every instruction that needs to be follow by the printer in order to have the piece printed: where to move, how fast to move and through which path.

The GCode is generated according to some parameters. They allow the user to define how fill is the piece, the pattern that is followed inside or the thickness that must have the piece borders.

Once generated the GCode there is only some things to take into account. One of them is the temperature that needs to have the extruder and platform. They may be found in the GCode already generated. When both elements reach the temperature needed the printing process is prepared to start.

The tools that have been used to print are:

1.4.2.1 Replicator G

This is the software that will be used in order to print with the oldest printer that was available in the department: a MakerBot Things-o-matic. Replicator G is software that will drive not only MakerBot Replicator and Thing-O-Matic but also is compatible with RepRap machine and also generic CNC machines.

You can give it a GCode or STL file to process, it can take the needed data from both. It is cross platform, easily installed and based on Arduino/Processing environments. In the Figure is shown how it seems:

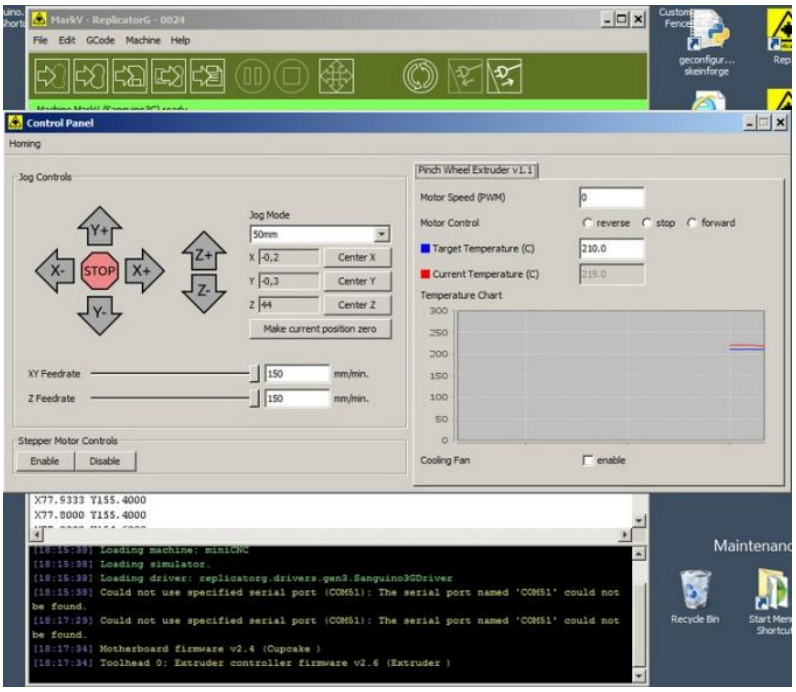

Figure 2. Replicator G Appearance

For printing the piece is necessary to select the printer, connect the program to the printer, fix temperature, generate the GCode and click the printing bottom. It is possible to know the time that the software estimates the piece to be printed.

1.4.2.2 Latest versions of Repetier

It is simple host software, which is compatible with most firmware around. In this case is used to control the Prusa Air 2. This is newer software but it works in a similar way as the Replicator G.

It is possible to add and position some designs stored in stl files. It may simulate all the pieces oriented and slice them together. For slicing, it is used the Slic3r slicer or the Skeinforge. In this case, contrary to the other software, after generating the GCode is possible to change some parameters. This was necessary in order to calibrate the printer.

This software was also used for controlling a third printer used. This was not available in the beginning of the printing process but, during the semester has been changed for new ones.

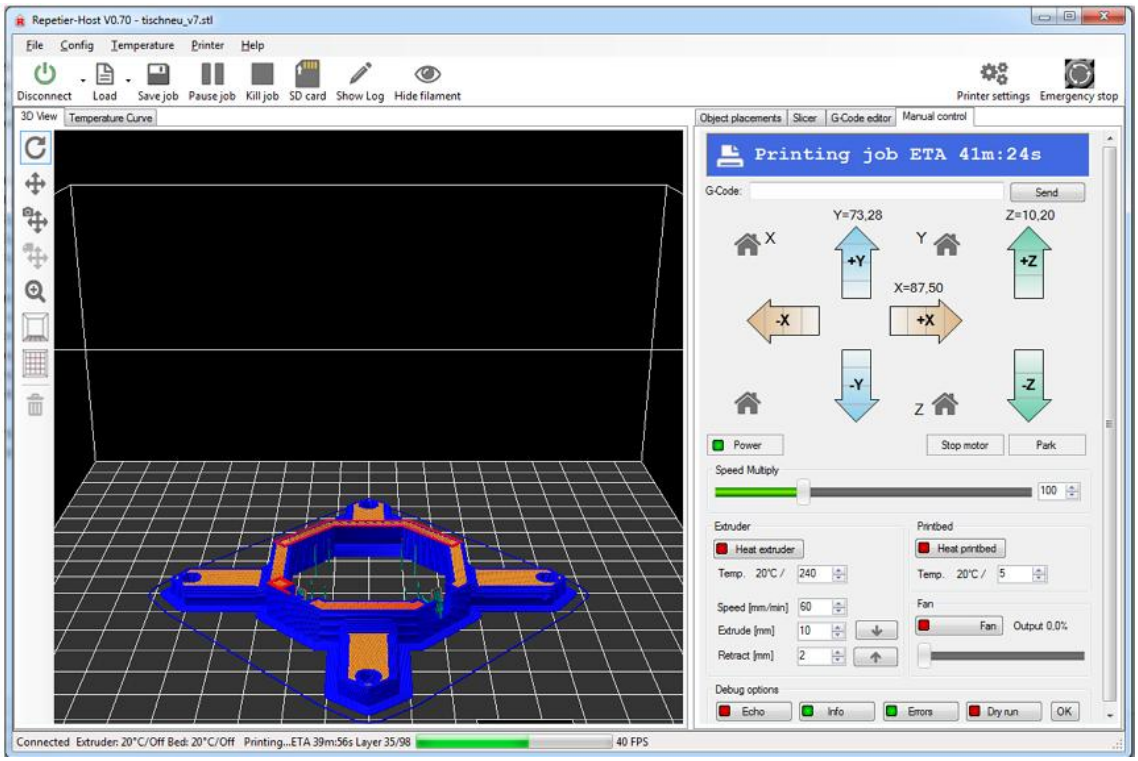

Figure 3. Repetier Apparience

1. LabVIEW

The third but most important section in this chapter of tools is LabVIEW. As it will be explain, it is the programmatic environment in which I will build the examples of Real Time Control.

It is designed to have similar work spaces for every kind of control and project. The advantage of this environment is that has specific modules design only to have Real Time systems and has many common codes already implemented.

It is important to know that this environment may be replaced by others with similar capacities such as the ones related to MathWorks. However, as it was also known how works the controller with LabVIEW, it was decided to use this one.

As long as the reader goes through the document, he will be able to read in detail some features that are particular from the environment. There will be a concrete description of two of the modules that allows the user to have Real Time control in two different manners.

1.4.3 RepRap

This is a web page mainly written by fans of 3D printing and designs and all the devices that allow every new users access to useful information. It is an online web page that has growth through the collaboration of specialists dispersed all around the world from varied fields such as mechanics, electronics & computers facilitated by this incredible network.

From this web page I have been able to find out every information described about the printers and some materials used in this 3D printing world.

1.4.4 yWorks

This is a free software tool that will be used to design every flow diagram that describes the functionality of the Real-Time project that has been implemented. Also it will be used to create the diagrams that show the user the types of data transfer used and the elements that compose the Real-Time system.

http://www.yworks.com/en/products/yfiles/yed/

1.4.5 Inkscape

This tool is a free downloadable software tool used to draw vectorial designs. In this project is used to design a custom control for the User Interface. This could also be done by using Photoshop or similar tools.

https://inkscape.org/es/

1.4.6 Microsoft Word and Adobe Reader

This is a known tool used to redact the memory. After having the whole project Report done the Adobe Reader was used to create save it in PDF format.

Chapter 2: Arm chosen & Print Out

Chapter 2: Arm chosen and Print out

2.1 Robotic arm developed

Before going through more complex concepts such as Real Time and the controller, it will be described the robotic arm with which the demos will be developed. As it have been said in the introduction the objective of this project is to introduce the reader into Real Time concepts and for doing this I have decided to take an already designed arm and print it out.

Although the design of the printable arm is not part of the project, the selection of which prototype would be controlled is it included in the workaround.

One of the first things to do in this project was the choice of the arm to control. First of all, it must be said that the final prototype needs to combine both, the simplicity of a 3D-approach and also the sophistication of a robotic arm.

As the idea of the project was to design the software application needed to control a real robotic arm, the functionality of the model chosen must be the more similar as possible as a real one. For this reason one of the requirements for the final decision includes six different degrees of freedom.

In addition, the design of the arm selected needs to be easily printable. In this part, there were some important facts to be considered for the choice:

a. The time spent in printing it out should not be long and it must be the cheapest as possible. For these causes, the size of the model should not be extremely big.

b. Each piece must be design for being printed. What is more, in this part it must be verify that there is no any piece that cannot be printed in a 3D printer. In this section are included some restrictions as:

➔ Avoid empty solids. In a 3D printer it is recommendable to have the piece relied on the base of the printer. However, it is possible that some parts need to be hollowed. In these cases, the best practice is to have the piece splitted in two or more parts and then stick them when building the solid.

➔ Avoid curvatures as much as possible. If curvatures are required, they can be done afterwards by smoothing over the piece manually.

➔ Try to have small pieces despite the fact that the arm is constituted by more components. Unavoidably there would be some problems when printing, if a small piece must be stopped there would be less plastic wasted.

c. Every piece of the arm must be available and easily accessible to be downloaded and therefore, printed.

In this case, finally was chosen a design that before this project was controlled with an Arduino. Moreover, it was designed by a student from the same university 2 years before.

http://www.thingiverse.com/thing:30163

The objective of the chapter is to allow the user to have a concrete idea of which is the appearance of the robotic arm and more precisely, how has been printed and mounted. In the following parts, there will be described all the processes needed to physically have the prototype mounted and prepared to be controlled.

2.2 Print out process

Although in the first chapter of the memory there is a section in which there have been explained the tools used in order to print the arm, in this part the idea is to specify the types of plastic that composed the arm and explain which printers have been used for having all the pieces done.

As it has been introduced in the first chapter there have been used three different printers. For being able to understand the differences between pieces printed in each of them in this section they are going to be introduced. In the printing process the 3D printers used are:

- MakerBot Thing-O-Matic: It was the second printer offered by MarkerBot. It was possible to buy it as a kit but also it was bought pre assembled. It accepts 3mm polymer filaments.

  It is based on an Arduino. This controller is the one that makes the servos move in order to follow the instructions that are saved in the GCode that is generated from the stl of the piece desired.

  This 3D printer is composed of a base that is able to get heated up and is where the plastic falls. This surface must be as smooth as possible, usually is used a mirror because of its price. Additionally, it must be easy to clean up, in this case it is not a mirror, and this printer has a metallic base made of aluminum. It also has a wood skeleton that supports all the mechanism and determines the maximum size of the piece that can be printed with that device.

- Prusa Air 2: this is a newer printer and was designed by Joseph Prusa who has simplified the building process of the printer. This device has a simple structure: it has the base and a vertical part in which four servo motors are located. These motors control the movement along X and Z axis.

  This structure is not made of wood as the MakerBot. It is made of plastic plates and the extruder moves along a metallic axis. Linear bearings are used for the Y Axis and T2.5 belts and aluminum pulleys for the motion transmission. Its base is made

of mirror and is bigger than the other one. For this reason, in this printer it is allowed to all pieces that are bigger that the others.

- Prusa i3 steel: this printer has been used for small amount of pieces in this project. It is very similar to the previous one and this may be explain because both are made of the same manufactory and also because this one is a combination of two previous ones in which we can find the Prusa air.

  The structure is very similar to the previous one but they have some differences, one of them is the fact that the plastic role of this one is not located in the upper part. Also, this one is made by 3mm steel plates which are cheaper but more heavy than the aluminous. For this reason this skeleton is more compact and tough than the previous one.

These two last printers are more precisely than the other and because of this, the most complex pieces, the ones that has curvature or small protrusions, are printed in these two last ones.

As it has been mentioned, there are two different plastics used to build all the pieces. These are special plastics used for 3D printers because of its low melting points and its price. They may be of different colors and thickness. These two kinds of plastic are:

- PLA: is a bio-degradable polymer that can be produced from lactic acid, which can be fermented from crops such as maize. This makes it an ideal candidate for use in certain energy rich, cash poor areas of the world.PLA is harder than ABS, melts at a lower temperature (around 180°C to 220°C), and has a glass transition temperature between 60-65 °C, so is potentially a very useful material

  It is dimensionally stable, so there is no need for a heated bed. You can get slightly higher quality surface finish with ABS over PLA, but on the whole PLA works better in the machine, requiring lower temperatures and giving stronger, more hard-wearing products. Its lower viscosity when molten means lower pressure in the melt chamber and hence a lower driving force. However, it is important to know that PLA filament can break when stored under stress.

- ABS: Acrylonitrile Butadiene Styrene is a commonly used thermoplastic as it is lightweight and can both be injection molded and extruded. It has better mechanical properties than HDPE and less brittle than PLA but handles higher temperatures better for applications such as extruder's and X-carriages setups without a fan.

  It is easier to buy and requires less force to extrude than PLA as it has a lower coefficient of friction. ABS is amorphous and therefore has no true melting point; however 230°C is the standard for printing. This makes its extrusion characteristics better for small parts, compared to PLA. The downside of ABS is that it has to be extruded at a higher temperature: Its glass transition temperature is ~105 °C.

It must be said that despite the fact there are two kinds of plastic; the building process of the arm was tedious. As both plastics were different, they have different dilatation coefficients so it the pieces do not fixed as desired.

After chosen out which prototype of arm will be used to demonstrate different types of control in this project, it must be printed. This is not a complex step. As in Thingverse the model is download in a format compatible with the printers, the only thing needed to print is time and plastic.

One of the advantages of the design chosen was that it was fully design to print it out. For this reason it has only a few complex pieces: basically the base.

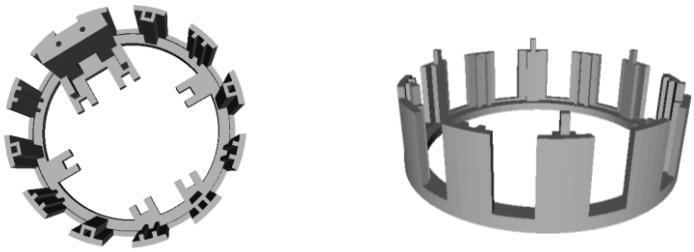

Figure 4. Base Piece Design

As it can be seen in the Figure 5, these two big pieces have two critical points:

1. The first one is the complexity of making both fix without any problem. For this reason they both were printed with the same printer and plastic. Also it was needed to maintain the same configuration in both printing processes.

2. The other critical point was the fact that both pieces were big but the problematic parts were printed at the end. In this way if something was wrong in that part all the piece must be rejected. Taking into account that the problematic parts were the junctions because of its thickness.

After having every piece of the arm printed, there two things left to do. The first one, is deciding the functionality of the arm, this means: which would be the extremely piece that defines which activities this arm is able to carry out.

The other thing left was mounting every piece together. In order to make this, it is necessary to make some tiny holes that allow joining the parts with screws. At this moment is time to introduce the servos in the correct place so that the building process moves on.

When the arm is almost finished, more specifically, when there is just left the last link or piece, then it can be seen that every servos is located in its final position and after verifying that every servo is moving correctly it is time to define the usefulness of the arm.

2.3 Arm Functionality

Before explaining the idea of which applications may simulate or be done by the robotic arm developed in this project, a general view of what can be done by real and 3D robotic arms will be carried out.

Nowadays, we are living in an innovation age where there are many works that are developed by robots and each month there is a new application example of task made by robotic devices. In industry dangerous task or jobs that need of strength are now developed by non-human devices. In this section we will see some of this task and more precisely how one of them will be adapted for a 3D arm.

Just to show some examples of robots that are used in industry and its functionality I have used the FANUC Web page to take some real photographs. FANUC is the number one in factory automation and in the last 60 years has been innovating in order to increase its productivity.

http://www.fanuc.eu/es/es

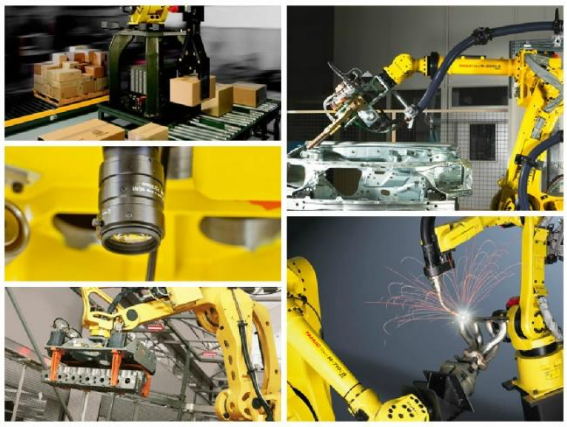

Figure 5. Different functionalities of robotic Arms

They are mainly dedicated to build robots. As it is shown in the Figure 6, there the typical utilities are: printing, palletized, welding, handing and vision. But they may combine two or more different tools in the same arm or use for another task such as writing or pushing object.

2.3.1 Robotic Arm functionality

As in the project from which the arm is, the only thing that is not maintained from the previous project is the arm functionality.

As the objective of this study is to demonstrate how powerful is the Real Time Control and give more than one example of controlling using this type of control, the idea was to allow the arm to do from simple to difficult tasks. However, with a clamp this functionality may seem the same as the one obtained with an Arduino.

The idea that was finally carried out was to develop a tool that allows the user changing the utility of this prototype. Taking the idea of one of the pictures from that web page, the objective was to design a piece that may be equivalent but made in 3D. For these reasons, the functionality of the arm will be defined with a simple but powerful piece that may allow the user to try many different tasks.

In the Figure 7 is shown how was made the transformation. In the FANUC tool they used it from repetitive task such as screw or label to control the movements in order to make the arm write something by making it holding a felt-tip pen.

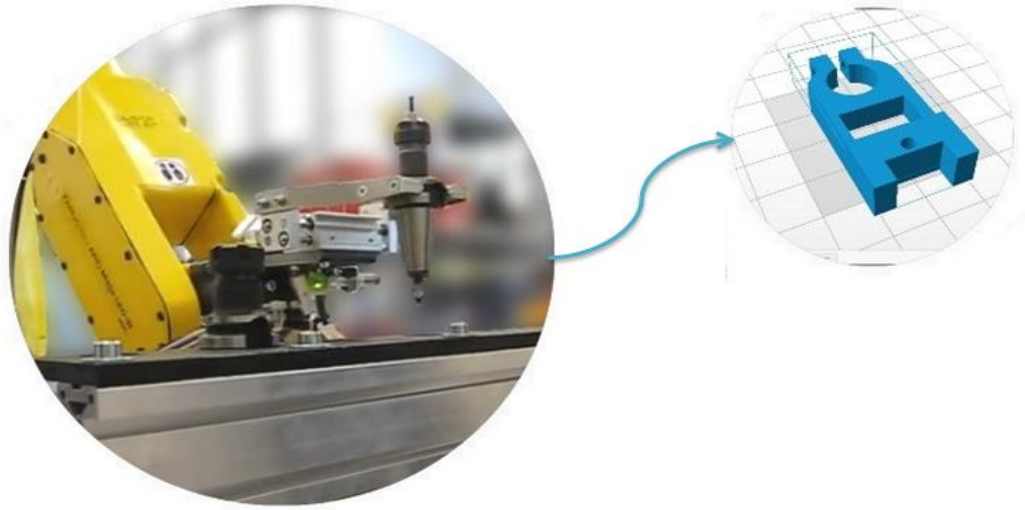

Figure 6. Functionality Arm-Design

The 3D piece show, which is the one that is printed already, may be used to pick any cylindrical tool such as a pen, or a punch to make the demonstrations. This piece was thought to complete the arm simulating some real robots that may be used not only to write, but also to make jobs that in industry would be dangerous and difficult to be developed by a human worker. The task I am thinking about are welding or picking some big elements with a suction pad.

This piece will be located at the end of the arm and despite the fact of its size; it is able to take different pens by a screw mechanism that allows holding the objects.

Chapter 3: Controller and electronic Hardware

Chapter 3: Controller and electronic Hardware

3.1 Servos description

For moving this robotic arm prototype the original servos chose by the designer of the arm will almost be maintained. With this, only one of the seven servos used is replaced by other known model.

In this final assembly there are only two different types of servos. The objective of this section is to explain the main features of both, not that specific as in a datasheet but the enough characteristics for understanding how they are going to be moved.

Firstly, a servo motor is an electric motor that runs using a control loop and requires feedback of some kind. Commonly, it contains a train of reduction gears and a potentiometer connected to the output shaft.

There are two different ways of controlling this king of actuators:

1. The first possibility is to control the position by using the output of the potentiometer. This method is based on a simple methodology. What is done is continually compare this position output to the commanded position from the control. Any difference is traduced into a signal that makes the servo move in the correct direction (either forwards or backwards).

   When the servo becomes nearer the position desired it reduce the velocity so when it reaches that specific position this signal becomes zero and the servo stops moving.

   If the servo position changes from that commanded, whether this is because the command changes, or because the servo is mechanically pushed from its set position, the error signal will re-appear and cause the motor to restore the servo output shaft to the position needed

   However the servos used in this project does not have this output of the potentiometer accessible. For this reason, they would be controlled by using a PWM.

2. The second possibility is control the servos by providing then a PWM signal. This will be the one used so a more specific explanation will be given. First of all, PWM stands for Pulse Width Modulation. It is a common technique that may be used in two different applications:

- On one hand, it is used in communication for encoding the amplitude of a signal right into a pulse width or duration of another signal that normally is used for transmission.

For this functionality, the main application fir which PWM is used is to send information in analog format. For a digital system, it is relatively easy to measure how hard a square wave.

However, if you do not have a Digital To Analog converter you cannot get information from an analog value as only you can detect if there is a specific voltage, 0 to 5 volts for example (digital values of 0 and 1), but this system is not able to measure an analog value .

However, the PWM in conjunction with a digital oscillator, a counter and an AND gate and door-step could easily implement an ADC. An example of circuit that may be used to create this kind of signal is shown in the Figure 8.

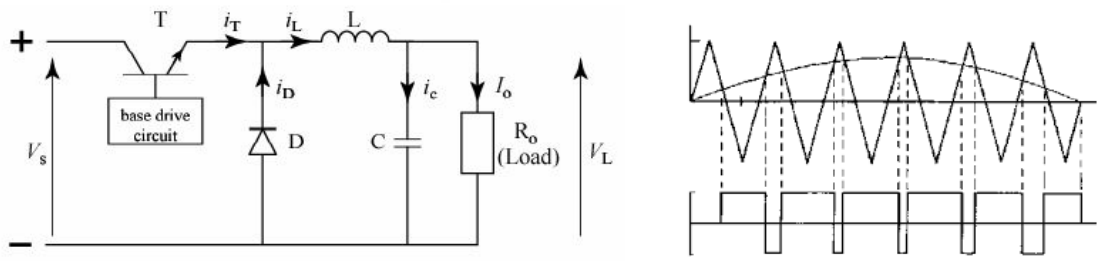

Figure 7. PWM Generation Circuit

The way it works is essentially controlling the amount of power, in the perspective of the voltage component, which is given to a device by cycling the on-and-off phases of a digital signal quickly and varying the width of the "on" phase or duty cycle. Duty cycle can be defined as the percentage of a total period in which the signal is active being the period the time it takes for a signal to complete an on-and-off cycle.

If we define D as the duty cycle, t the time the signal is active and T the total period. The duty cycle may be calculated as:

$$D = \frac{t}{T}\ x100$$

Equation 1. Duty Cycle

To the device, this would appear as a steady power input with an average voltage value, which is the result of the percentage of the on time. The duty cycle is expressed as the percentage of being fully (100%) on.

- The other application and its main purpose is actually to control the power that is supplied to various types of electrical devices, most especially to inertial loads such as AC/DC motors. For illustrating this second use the Figure 9 shows an example with one of the servomotors that will be used in this project and how its position change depending on the pulse with the user fix.

In the practical part, the way to move the robotic arm that has been printed is by generating 7 different PWM signals. Depending on the position desired by the user, the project will allow by means of a user interface to vary the position of the servos by changing the value of the pulse with.

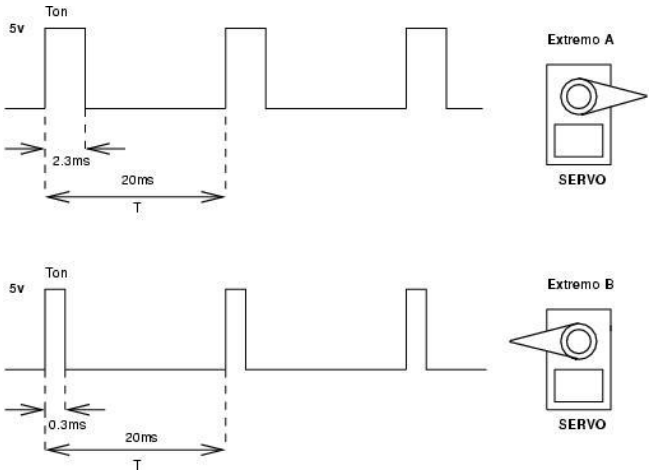

Figure 8. Servo Movement controlled by PWM Signal

Once it is clear the control mode with which the servos in this project are controlled, it will be explain the main characteristics of these two models used.

3.1.1 Servos used during the project

The first thing that must be said regarding to servo motors is that most of them have a three wire interface. Commonly, they have a red wire for the input voltage, a black wire for ground and a white or orange wire for the control signal.

Before taking into account the characteristics of the servos used, it is necessary to know how they would be feed. In the beginning the idea was to take advantage of the myRIO capacity to supply voltage signals. However, this idea was discard because of the possibility the controller does not have enough capacity. What is more, it was tested and with the myRIO just four motors may be fed.

Taking into account that the arm needs to have seven servomotors feed in order to be able to move the way desired, the final decision was to use an external power supply in order to be sure that there is no problem of supplying. However, if the arm needs to be moved the supply may be done with the controller with some additional supply such as a battery.

After knowing how they are feed, the objective of this section is to review the important features of the motors. In this part, it is explained not only the ones used but also the other with which the arm has been tested too.

3.1.1.1 Servos used

Futaba S3003

The first type of actuator used to move the robotic arm is the Futaba S3003. This first type is the biggest one. Its size is 40x20x36mm volume and it can be move 120º range from one extreme to the other. It needs 5-7, 5 Volts to work correctly.

As it may be seen in the datasheet, this is a servo motor that is control by a pulse with modulation that may vary from 500 to 3000 micro seconds and it will be used for the base and the first two joints. As it is bigger, it's with is bigger too so it will help the arm to be more static.

http://www.servodatabase.com/servo/futaba/s3003

In order to have a bigger range in the base, there will be two of this servos located in a way that both range does not coincide. For this reason the arm will be able to move double its base position. This is the only model finally maintain after the test because is cheap and it fulfill all the requirements.

Tower Pro SG90

This actuator is known as micro servo because of its size and weight. At first, the idea was to maintain the one specified by the designer of the arm but that other type did not work correctly. This micro servo is offered by Arduino and its weight is only 9g.

http://www.micropik.com/PDF/SG90Servo.pdf

It fits the arm because he dimensions are very similar to the order, from the specification sheet: 22.2 x 11.8 x 31 mm and its movement range is 180º. They are also controlled with a PWM signal and needs 5V to work correctly.

Tower Pro MG995

This is one of the 2 models that have been changed from the one that were selected by the arm creator. This servo was chosen because of its higher par and its lower price regarding to the one previously chosen.

http://www.servodatabase.com/servo/towerpro/mg995

As it can be seen in the datasheet, this servo also is feed with 5V its movement range is only 60 degrees and its torque goes up to 9.5 kg-cm.

3.1.1.2 Servos tested but not used

Tower Pro MG90

The second kind of servo that will be used is the Tower Pro MG90. This is a smaller servo that will be very useful in the latest parts because of its small size and weight. Its data is: 23x11x24mm and only 9,5g. For this reason it will be used in the wrist to move the arm.

One of the advantages of this servo is that despite its small dimensions it is able to move 1,5kg/cm and has a range of 120º. This makes possible some applications for which the arm will be used.

http://www.servodatabase.com/servo/towerpro/mg90

Finally, this servo needs to be feed with a voltage that may vary from 3 to 5 Volts and has a neutral pulse with of 1500 micro seconds.

Dinamixel xl -320

This servo was tested for various reasons. The main one was its modularity, this means that although in this case was tested the smaller one, this servos are design to work the same way with the bigger ones.

http://www.trossenrobotics.com/dynamixel-xl-320-robot-actuator

In this case, they were finally not used because of the price and the difficulty of needing 7.4 V to feed them.

3.2 Connection specifications

After knowing the servos that will be used and having every piece printed, it is time to show the connections of the arm. In this section, it will be shown not only how to take every cable coming from the servo motors to the arm base but also the pins used in the controller for every servo and how it is needed to locate the cables to allow the maximum efficiency of the arm and the largest movements.

For this part some images from the Final Degree Project made by Antonio Castro are used. Some connections are maintained but some others have been changed in order to use newer materials available in electronics shops.

http://biblioteca.uc3m.es/uhtbin/cgisirsi/x/UC3M/0/5?

It is also important to take into account that in this case, there are only three cables for each servo contrary to the arm built by Antonio. In that case he controls the servos by taking an additional cable from the servo to now the position by using the potentiometer output. In this case, at is has been explained it will be used the signal cable to fix the position using PWM signals.

Finally it must be mentioned that for making the instructions easer to follow, there will be maintained the nomenclature of every am and servo in the arm that is used in Antonio Project. Just to clarify each piece and name is:

- Part 0: This is the base of the arm. It is fixed to a wooden piece that allows the arm to be stable. This piece is composed by 3 big pieces that has been mentioned in the printing section and there it is inserted the servo motor A1

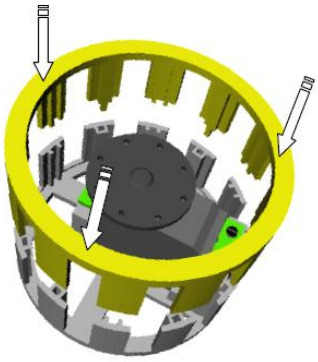

Figure 9. Robotic Arm: Part 0

- Part 1: This second part is located on the top of the first one and it is composed by 4 pieces. It has two servo motors inserted on it. One of them allows the arm to move 360º if it is correctly fix with servo A1. This second servo is the A2 and the third one, located in one of the sides, is used to move part 2 and it will be named A3.

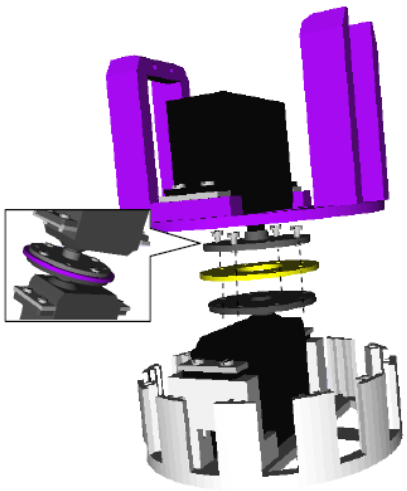

Figure 10. Robotic Arm: Part 1

- Part 2: The third part is used to move the main arm; it is composed by 14 pieces and is moved by servo A3. There is no servo motor located on it but in this piece there will be cables from the upper motors fixed on its intermediate part.

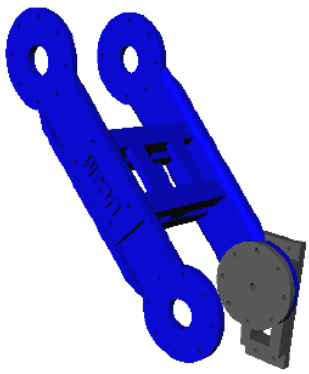

Figure 11. Robotic Arm: Part 2

- Part 3: this piece is similar to the other one but it does have two servo motors located on it, one of them, it will be called A4 and is used to move this part with respect to part 2 and the other one, known as A5, is used to move part 4. Part 3 is built by using eleven pieces.

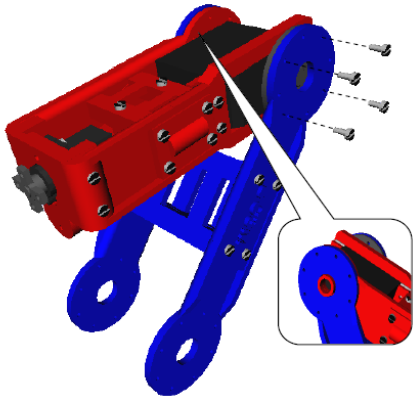

Figure 12. Robotic Arm: Part 3

- Part 4: This part is used to provide one of the 3 DOF (Degree Of Freedom) to the end of the arm. It is composed by three small pieces and servo motor A6 will be located on it.

Figure 13. Robotic Arm: Part 4

- Part 5: This is the last part that is taken from the original device. It is composed by 3 pieces and to this part is connected also the tool design. The servo left is located here and will be named as A7.

Figure 14. Robotic Arm: Part 5

3.2.1 Cable connections

This part will be shown the how to connect the cables from the servo motors along the arm. For this reason, it will be shown mainly with images:

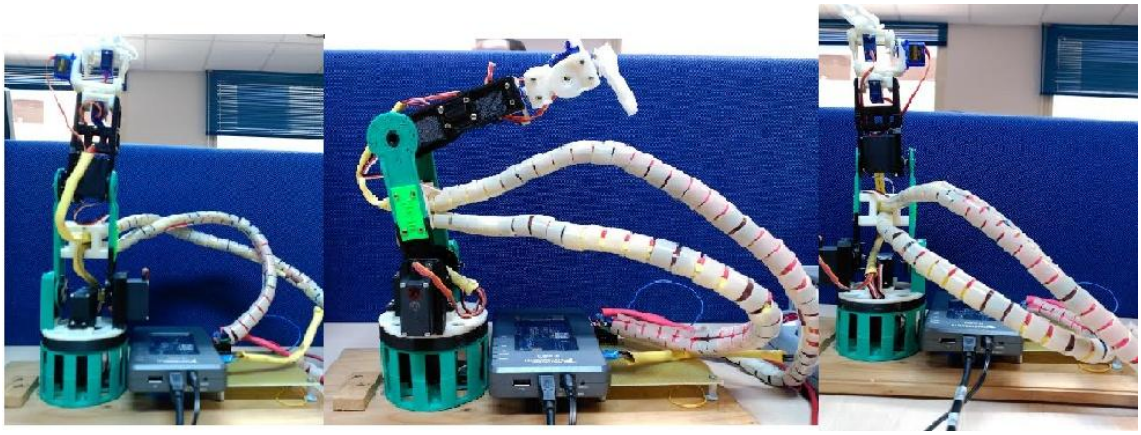

Figure 15. Cable connection from Servos

The Figure 16 show the appearance of the complete arm mounted. For connecting every cable there are some materials needed:

- Unipolar cables (de Arduino): these cables are used in order to take the signal from the servo to the controllers. They are cables that already have the connector when bought it so it is faster to build these connections.

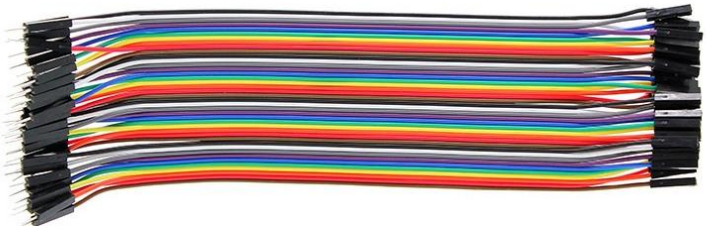

Figure 16. Cables used for connections

- Perforated plate: It is used in order to feed every servo motor together with one source. With this perforated plate every positive and ground supply cables are welded together. This is shown in Figure 18.

- Lined heat shrink tube: this special material allows to group together in the same tube as cables as needed. With this, only are shown 4 tubes and not 3 small cables for each motor (total of 21) and is weirder that a cable may be disconnected or be fastened with any piece of the arm.

- Heater: This tool will be used in order to heat the lined heat shrink tube and make it tighten itself.

- Screwdriver: it would be used to rebuild some parts of the arm in order to locate the cables in between.

This step is very arduous because there are some problematic parts that require unbuilding some parts of the arm. These problematic ones will be shown now in order to allow the user to build this prototype if needed.

1- The first thing is to measure approximately the length needed to have the cables connected to the controller but with enough play. In the table it is shown the measurements of each of the cables connected.

Servo motor	A1	A2	A3	A4	A5	A6	A7
Length (cm)	28	28	28	48	37	57	57

Table 1. Cable Length from each servo to the controller

2- The cables coming from the three smaller servos must be located inside the arm parts. It is needed to unscrew the intermediate part so that it may be taken off and then locate the cables there.

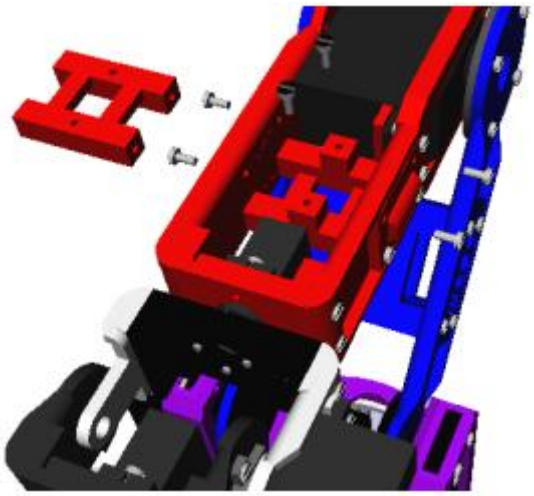

Figure 17. Cable Arrangement through the Arm

3- It is also important to know that the groups off cables that are connected in the controller are divided in two different parts in order to have the maximum movement freedom. For these reason there are 3 servos connected together and the other two separately. It can be seen in the Figure 16 (la del principio)

3.2.2 Controller connections

After having all the cables coming from the servos down to the base, it is the time to describe how they are connected to the controller. Although the controller is not jet explained, it is important to take into account that the one used is a myRIO 1900.

The myRIO is a device that has been design only for student use. It is composed by a micro controller and a PFGA. The idea behind is to allow students getting used to Real Time and FPGA control applications and also be familiarize with applications of Data Acquisition and Conditioning.

http://www.ni.com/pdf/manuals/376047a.pdf

This section is focused on the I/O available in the controller, but this device has also an available accelerometer and Audio I/O. From the datasheet it may be seen that there are not only analog and digital I/O but also preconfigured PWM I/O. However, as the signals that are used to control the arm are totally generated from zero, the I/O use are the digital ones from connectors A and B. There are two kinds of cables on this:

1. Supply cables: these are 14 in total, one positive and one ground for each servo. In this case it was decided to use an extern supply in which the seven motors were fed. For doing this, all the supply cables will be weld in a perforated plate.

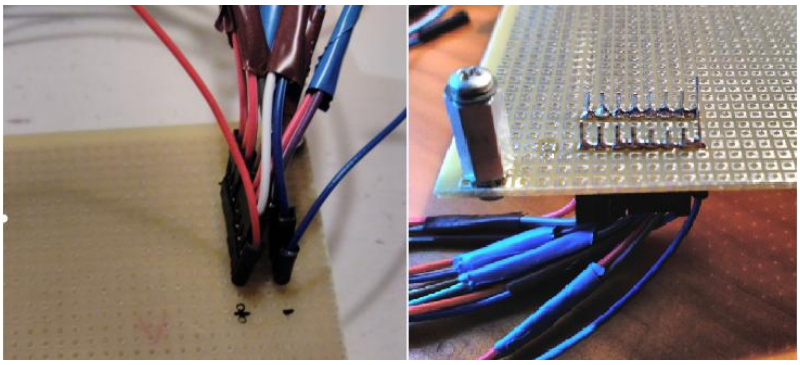

Figure 18. Welded perforated plate

As it has been said before, in the beginning the idea was to feed them with the myRIO but, afterwards it was decided to connect this way in order to avoid bigger problems and unpredictable servos movements.

2. Signal cables: these are connected to the DO signals of the myRIO. Depending on the sample project in one pin or others. In this part it is illustrated the connection of the servo signal cables for the RT process which is the most complex one.

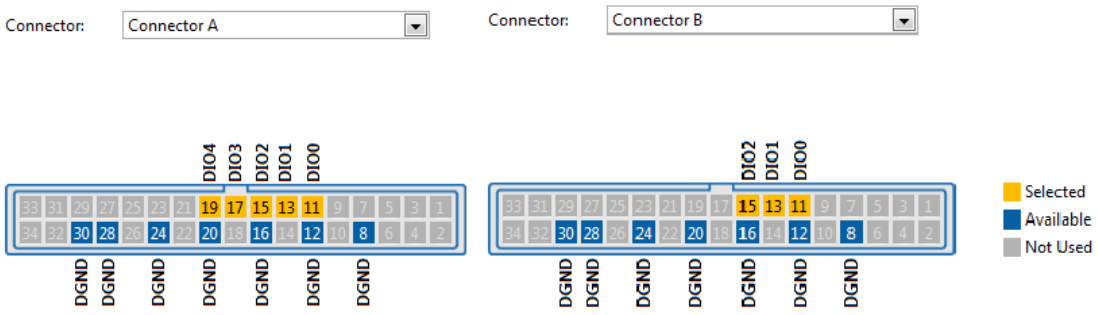

Figure 19. Controller connections

Chapter 4: Real Time Concept and Environment

Chapter 4: Real Time Concept and Environment

4.1. Real Time Concept

In the control field there are some different levels depending on various factors that defines how complex does the system may be, the accuracy and response time of the control process.

Depending on these factors there may be found different types of control. In this chapter it will be presented one of the most complex concepts when talking about control: Real Time concept.

Real Time concept is based on synchronous languages. The idea is to have a system that will execute its tasks in a certain time, for sure. When designing a Real Time System there are some key aspects to have in mind:

1. The system will have a code divided in some parts that need to be executed in a specified time. In order this to happen, first the user needs to fix the time that the code has to be executed.

2. In a complex system there will be more than one task. Each task will have its own priority and the objective when designing the system is to define the priority with which the code will be executed.

3. Not every function and not in every target it can be implemented this kind of codes. For example, a common RT Application is a closed-loop control that would benefit from using RT because it must produce an output based on input values in a guaranteed time frame. This will be the case of a PID control system or an ECU (Engine Control Unit) modeling that may be simulated with a closed-loop.

   However, a general purpose Operating System cannot guarantee that a specific application will be executed within a fixed time. For example, in Windows, the task execution depends on many factors: other programs running in the background such as screen saver or antivirus.

For this reason, it is important to learn which applications need to run in a Real Time Operation System (RTOS) or can be considering General Purpose Applications. In this second group there can be included Data Acquisition, Offline Analysis or instrument control whether it is advisable to use RTOS for time critical decisions or extended run time applications.

In this chapter first will be explained the base of the Real Time Control. Afterwards, there will be explained the main features and the main environments that may be used. Finally, the environment chosen for this project, LabVIEW, will be better introduced.

One of the main factors that may introduce delay in the control system is the presence of more software but the one needed to control. As it will be explained, one of the most important advantages that are introduced when talking about Real Time concept is that there are used targets that do not need any additional software installed, such as Windows.

4.1.1 Real Time Base

As it has been said, Real Time is based on synchronous languages. A synchronous language is a computer programming language optimized for programming reactive systems. Computer systems can be sorted in three main classes:

1. Transformational Systems: are used to take some inputs and deliver them as outputs after a processing procedure. Once executed, they stop until they are asked to execute again. An example of this system may be a compiler.

2. Interactive Systems: interact continuously with the environment. Its state and outputs depend on its inputs, for example the web.

3. Reactive Systems: they are computer systems in which iterations with the environment are prominent aspect; this means that they are periodically executed. They must therefore react to stimuli from the environment within strict time bounds when they are running. For this reason they are often also called real-time systems, and are found often in embedded systems

The problem of specifying, programming and verifying Real Time reactive systems, together with the definition of appropriate development environments, is still an important research problem that has been doing some important progress, the following being widely accepted.

There exist appropriate design methods, programming languages and environments for transformational systems. In the case of reactive systems, like Real Time process controllers the need for formal verification methods and tools is even more crucial than for transformational ones because strong reliability requirements are associated with them.

Synchronous programming is also known as SRP that stands for Synchronous Reactive Programming. The base of SRP is the synchronous abstraction; the idea is to make the same abstraction in programming that the one done it in digital circuits. They are based in synchrony hypothesis which states that the reaction time of the system is zero. From an external point of view, it means that outputs are produced simultaneously with the inputs, which is clearly unimplementable; however, a synchronous system works fine provided it reacts sufficiently fast.

A synchronous circuit is characterized by its high-level abstraction timing. The synchronous program reacts to its environment in a sequence of ticks, and computations within a tick are assumed to be instantaneous. It computes instantaneously output values and the new values of its memory cells from the input values and the values that are currently in the memory cells.

Synchronous programming languages were born in France in the 1980s. The first SRPs were Esterel, Lustre and Signal but afterwards, there appeared some more new ones. Nowadays, the most important ones in this field are:

- Argos: is a set of operations that allow combining Boolean Mealy machines in a compositional way. It takes its origin in State charts but with Argos operators, one can build only a subset of State charts, those that do not use multi-level arrows.

- Lustre: was created in the 1980s. In has been progressing to a more practical, industrial use as a core language of the SCADE environment. It is used for critical control software in aircraft, helicopters and nuclear power plants.

- Esterel: the main functionality that characterizes this synchronous programming language is the fact that it allows the simple expression of parallelism and, for this reason; it is well suited for control model designs.

  To take a concrete example, the Esterel statement “every 60 second emit minute” specifies that the signal “minute” is exactly synchronous with the 60-th occurrence of the signal “second”.

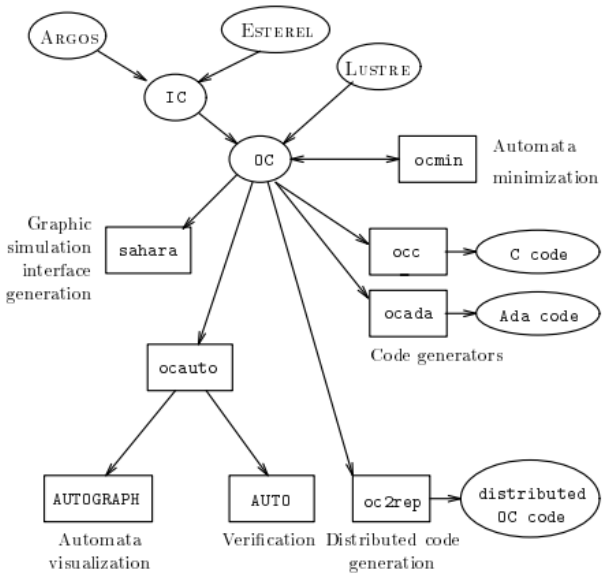

Figure 20. Relationship between programming languages

In the Figure 21 it is illustrate a common environment of these three first programming languages and how they used to work together.

- LabVIEW: is a system-design platform and development environment for visual programming language. It is a graphical language which use is slightly different form the ones that are used in MATLAB/Simulink that are focuses on low lever aspects referring to the use of FPGA.

  LabVIEW is an inherently concurrent language, so it is very easy to program multiple tasks that are performed in parallel by means of multithreading. This is, for instance, easily done by drawing two or more parallel while loops. This is a great benefit for test system automation, where it is common practice to run processes like test sequencing, data recording, and hardware interfacing in parallel.

- SIGNAL: was the first designed for signal processing applications in the beginning of the 1980s. Since then, it has adopted a dataflow and block diagram style with array and sliding windows operators.

  It is a dataflow-oriented synchronous language enabling multi-clock specifications. This programming language allows to specify and application, design an architecture or define in detail components down to RTOS.

- Atom: was created in 2007 by Thomas Hawkins. It is a DSL in Haskell for hard Real Time embedded programming. It is a programming language design for embedded

systems that offers compile-time task and generates code deterministically. Moreover, it eliminates the need for using mutex locks.

Mutex stands for Mutual Execution. In programming, a mutex is a program object that allows multiple program threads to share the same resource, but not simultaneously, such as file access. When a program starts, a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource. Thos means that it is set to unlock when the data is no longer needed or the routine is finished.

- SyncCharts: allows specifying the reactive behavior, and the synchronous programming, of applications. SyncCharts is the name of the model and it is also used as an instance. It inherits many features from Argos.

At a more fundamental level, the synchronous abstraction eliminates the non-determinism resulting from the interleaving of concurrent behaviors. This allows deterministic semantics, therefore making synchronous programs amenable to formal analysis, verification and certified code generation, and usable as formal specification formalisms.

In contrast, in the asynchronous model of computation on a sequential processor, the statement “a||b” can be either implemented as "a;b" or as "b;a". This is known as the interleaving-based non determinism. The drawback is that it intrinsically forbids deterministic semantics which makes formal reasoning such as analysis and verification more complex.

CASE Tools

CASE stands for Computer Aided Software Engineering. These tools represent a combination of programs which objective is increasing the performance and reliability in software development. The idea behind is to reduce the amount of time and resources needed for the system development.

In order to reach this entire objective, a CASE tool uses programming languages based on graphical tools. They provide a high level abstraction that helps the user to define the functionality desired in a more intuitively way. Additionally, these tools automate all the writing processes: they used code in textual format and also they automate the error checking and adjust all so that they all fulfill the security laws.

These Tools may be work in different fields. There exist some CASE tools oriented only to the design of user interfaces; others are focus on simulation o programming embedded hardware devices.

In order to be more specific and have a concrete idea of which is CASE Tools it may be mentioned some ones such as MATLAB/SIMULINK, LabVIEW and every tools that belongs to IBM Rational.

- MATLAB and SIMULINK are tools that normally are used in embedded control applications and one of the advantages that can be found is the big amount of additional software tools that helps the designer when creating the application. This additional software packs are known as toolboxes.

- LabVIEW has large of industrial hardware devices that are design only to control with this tool. Although the controller used in this project is controlled by LabVIEW it was design for students.

- IBM Rational is able to support large and different kinds of applications, from embedded control systems to user interfaces located in a PC. However, it does not have additional toolboxes.

4.2. Real Time Environments

After having a general view about RT concept, its origin and some tools used, the objective of this section is to provide a look over the most important environments in the market that also allow this kind of control by offering hardware compatible with it. These environments are the actual solutions used for RCP in sectors such as aeronautic or industrial one.

The idea is to show briefly some of the possibilities that are available for these graphical tools, its advantages and some disadvantages too. Although cRIO and myRIO are not included in this section, they will be explained in the next one.

4.2.1. Hardware and Toolkits that may be used with MATLAB/SIMULINK

In this first part it is explained the way to work with MATLAB/SIMULINK is the one shown in the Figure 22. First, it is necessary to create a model of the plant that will be used in the application, which is the one containing every element to control. Afterwards, a control algorithm is created and then the RCP Hardware is programmed and tested before applicate everything in the real plant.

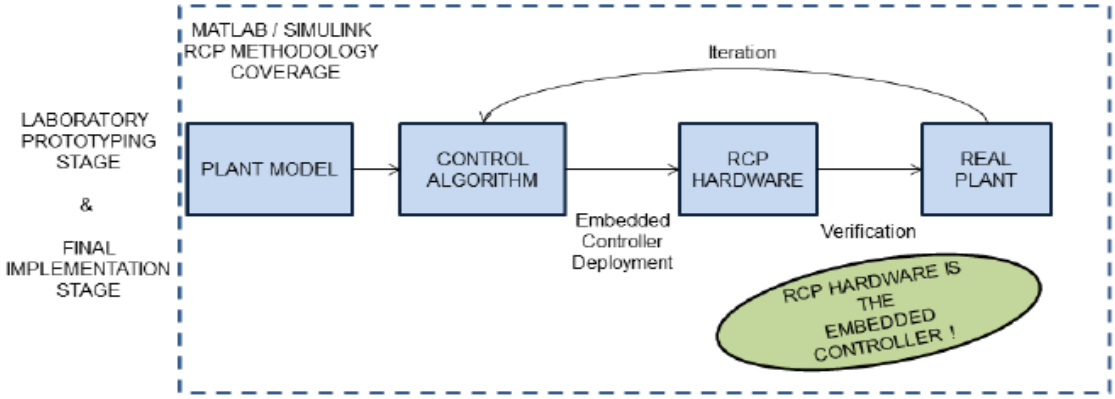

Figure 21. Working Methodology when using MATLAB/SIMULINK

4.2.1.1 xPC Target

xPC is a toolbox created by MathWorks that is used to control hardware compatible with this platform by using MATLAB/SIMULINK. It is focus on compatible processors Intel x86 and x64. This means that there are many computers that may be used with this toolbox. If then, these PCs are needed to be used in a control application it is necessary to add them some additional targets to include digital and analog I/O.

These I/O targets are designed in the way that they may be placed one up to the other. In this configuration they are connected to a PCI interface that stands for Peripheral Component Interconnect. For this reason, the appearance of these RCP Systems is used to be a large vertical tower.

The way to work with this RCP system consists in using Simulink to create a visual model of the control system that is desired. Simulink allows the user to modify parameters of the controller as well as to purify the performance of the system of control. Simulink will compile this final model and will load it in the working station. This working station needs to have installed the RT software of xPC Target; this in the xPC target where the model of control will be executed. The communication between both computers is done via Ethernet.

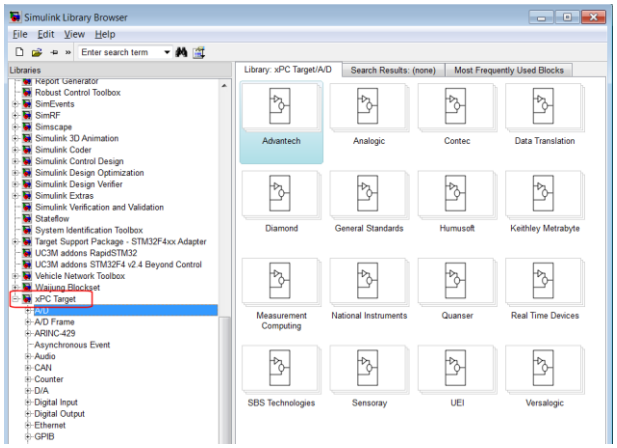

Figure 22. Simulink Model Components

When working within this environment is necessary to know that in order to have a computer working in RT it is needed a host PC where the Simulink model is developed and from where data is received and sent from and to the Target. As it is shown in the Figure, it is also possible to have the system working completely alone but
For this issue, there is a box from MathWorks that allows running the model as executable; this box is known as xPC Target Embedded Option.

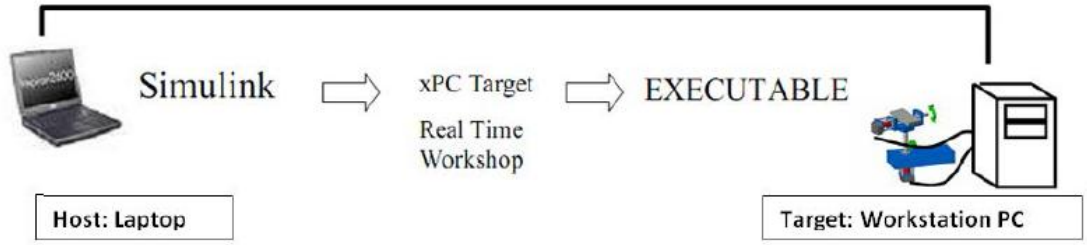

Figure 23. xPC Target Embedded Option Diagram

The process to create a digital embedded controller based on a PC of this type is made by using an intermediate element that allows the target two different things: on one hand to know how the system is configured and on the other hand to be able to communicate with the different I/O that in this case are connected directly to the xPC target. This intermediate element may be a special SD card (in the first case the system will be a Single Board Computer (SBC)) or a compact CD ROM that contains all the information.

The main disadvantage of this architecture is that the size and energy use are not designed to be embedded control systems. Moreover, the shortage of these intermediate builders in industry makes the price of an xPC Target based system being very high. For this reason this is not the environment that is used for education or in any university laboratory.

Within the large programming applications made using the xPC Target toolbox there is no way to control different RT tasks. One of the problems that will be found here is that every I/O needed is added with PCI Targets. However, this Toolbox is used in industry for controlling exoskeletons or mechatronic robots. They are mainly control task for electronic systems, motors and electric storage.

4.2.1.2 dSPACE

dSPACE was created in 1988 and it is focus on providing products, not only software but also hardware for Rapid Control Prototyping (RCP). The software and hardware products that dSPACE® offers are integrate in Matlab/Simulink as additional tools in the form of specific toolboxes, largely oriented to control. They are used in industrial fields such as the automotive, aerospace, medical devices and production lines. dSPACE Systems may be splitted in two different tendencies:

1. The first one is focused on testing the different control models by using an intermediate RT Targets that are only used to prototyping. When creating the visual model in Simulink, there are used some simulated models which are known as PIL that stands for Processor-in-the-loop.

2. The other one is oriented to test the RT model and the plant simulation, all executed in a RT simulation model known as HIL that stand for Hardware-in-the-loop. In this model there is a virtual industry plant and a RT controller , both running on Real-Time hardware, with sampling times of the order of microseconds

dSPACE hardware is design to work with HIL. It has a high computational power mainframe. Most of the RCP from dSPACE systems use IBM PowerPC processors that will be briefly explained in the following section. The FPGA is only used to treat high-frequency digital signals by using expansion modules I / O.

The processor that executes the Simulink model is located in a PCI target, running under the Real-Time Operating System QNX. Similarly to the case of xPC Targets, these cards are used as expansion parts, providing I / O capabilities to PC. In the Figure 24 is shown an example of one of the cards that can be included in this group of devices.

Figure 24. PCI Card used as expansion I/O to PC

This is the DS1104 card. It also has a microcontroller from Texas Instruments with a processor unit that allows 16-bit floating-point calculation, operating at a clock frequency of 20 MHz, with 32KB of ROM program memory and 8 KB of RAM. It provides 7 PWM outputs, 4 inputs PWM 1 SPI port and 14 digital I / O further. Its estimated cost is 950€ or 4800€ together with its toolboxes for Simulink.

Apart from this card, there are others which have larger capacities. On them each part is separated; one example of this may be a card with a RT processor of 1GHz which

has additionally expansion cards that provide I / O module connectors. There is a wide portfolio adapting to the complex or specific needs that are required.

Regarding the RCP systems whose operation is completely independent from a PC, dSPACE offers a hardware called Micro-Autobox in smaller format that uses the IBM PowerPC processor with 900 MHz, or a slightly larger format with Intel ATOM processor size. The version with Intel processor may not work in Real-Time with the QNX operating system.

It should be noted that this is a RCP system that is slightly oriented to be used in automotive applications. For this reason, its CAN and LIN interfaces are supported for car control calibration protocol standardized by bus or through other communication interfaces, in that case being referred Calibration Protocol extended (XCP).

To provide I/O capabilities to these RCP will be providing the appropriate I/O cards. In Real-time Micro-AutoBox systems an expansion card I/O can only be installed. The price for universities, together with its toolboxes Simulink and card I / O, is around 12,500€ per unit. The Figure shows how these expansion cards Micro-AutoBox system are connected.

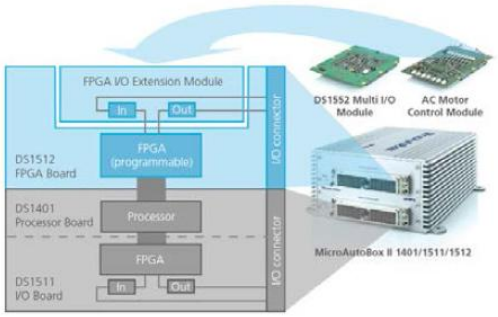

Figure 25. Micro-AutoBox system

Each I/O available functionality has its own toolbox that is sold separately. It can be found a generic toolbox for managing Real-time processor, controlling the RT operating system QNX tasks and associated toolboxes they will be needed to cards expansion.
In the Figure it is shown the graphical programming blocks available for the CAN bus of any RCP belonging to dSPACE systems. There are set of blocks called RTI (Real-Time Interfaces) CAN.

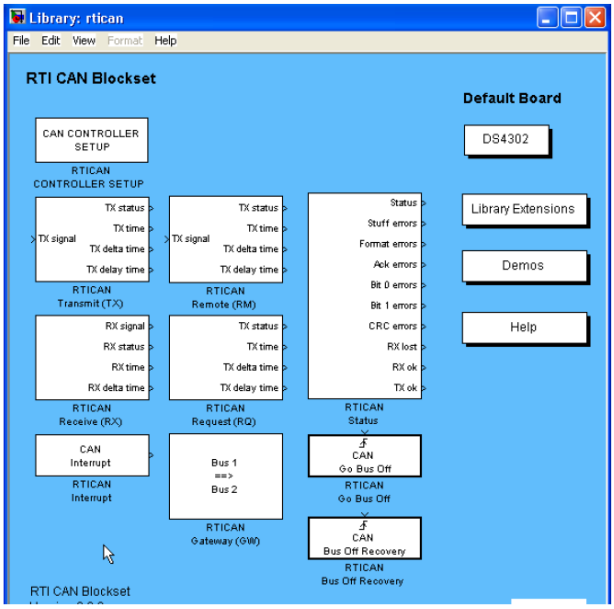

Figure 26. dSpace Real Time Interface for CAN

4.2.3. IBM Rational

The family of IBM Rational software covers a wide spectrum of applications' making heavy use of UML and SysML languages. There are programs within this family for creating user interfaces and applications management and query databases for PC computers, primarily through language that generates graphics based on the textual source code in C, C ++, ADA or Java.

Other programs belonging to Rational family are used to carry out simulations of multidomain systems. All programs that belong to Rational family can be integrated with each other, like a single model-based environment is involved, complementing its functionality.

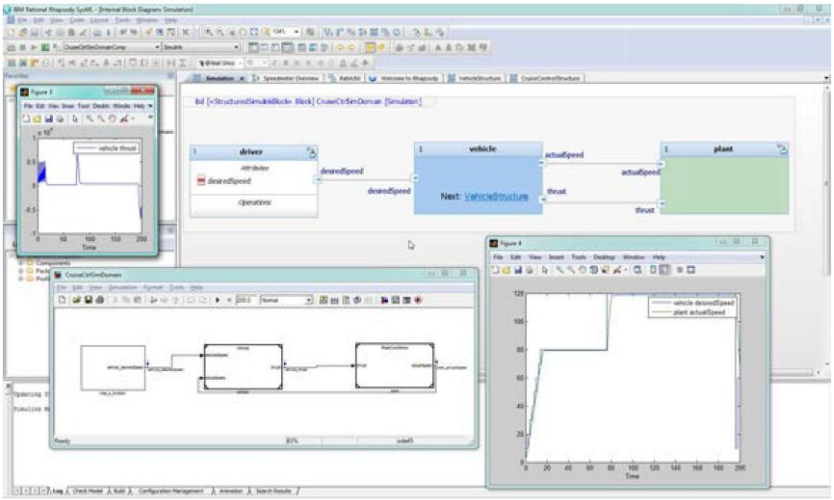

Figure 27. Model created using IBM Rational

Both Simulink and Rhapsody features are so similar that the software can be related to each other, transferring graphics from Rhapsody to Simulink models and vice versa,

although Simulink is not based on UML/SysML. In the Figure 28 it is illustrated a Rhapsody model imported into Simulink. One of the relevant when using multiple MBD software simulation for the same reasons is the comparison of results.

4.3. CompactRIO & myRIO

RIO stands for Reconfigurable Input Output. CompactRIO and myRIO are reconfigurable embedded controllers designed for control, data acquisition and data processing. The hardware architecture of cRIO and myRIO systems is based on FPGA. This FPGA acts as means of input/output and will be explained in the following chapter.

This architecture also has a processor (microcontroller) where the Real Time Operating System (RTOS) works; in this case the RTOS is VxWorks.

One of the things that is needed to be in mind of the reader is that the processor does not have any input/output configured at first. In spite of having peripherals, in order to have available I/O it is necessary to communicate to the FPGA. The FPGA transfer the data from the I/O interfaces that have been synthesized in the silicon chip, or are connected to the FPGA trough external expansion modules.

In the Figure 29 is illustrated this concept in a simplified way.

Figure 28. LabVIEW Reconfigurable I/O Arquitecture

As mentioned before, these systems are programmed with graphical programming tools created by National Instruments. In the following section this environment will be better explained but in order to illustrate a bit how does it works it has been created a simple Data Acquisition example.

There are two different ways for programming in LabVIEW. In this small example first I will illustrate how to make some quick checking of how a system or specific device works using what is called VI Express and afterwards, it will be show as a block diagram completely building in a lower level of programming.

At first, it has been built a simple example that takes data from a device and plots it in a graph. The VI will look like shown in the Figure.

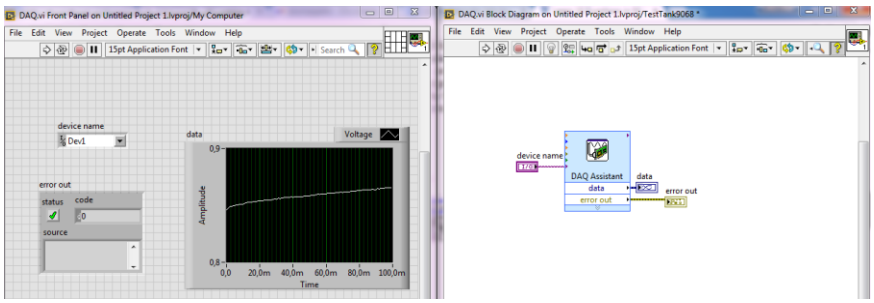

Figure 29. LabVIEW Example using VI Express

As Express Vis are not normally used as part of the code of real applications for its complexity and unknown functionality, the correct way of developing this application would be by using simple Vis and design the application from zero. For doing this issue, the user must use the low levels Vis and write the code that corresponds to the functionality desired.

In order to have an idea of how different would the two Vis be, I have developed the same example as before but with simple Vis. In this case, the advantage is that anyone that wants to understand how the project works, can. This is possible before each blocks its own task and by following the data flow it may be understood the functionality of the VI.

However, being able to design every part of the application code requires a high level of knowledge of LabVIEW Environment and there are not many users that are able to design the whole application from zero.

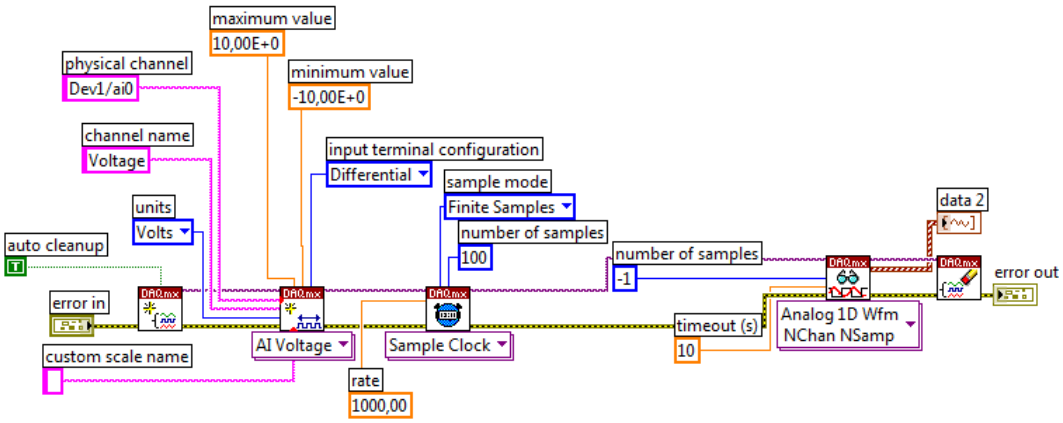

Figure 30. LabVIEW Example for Data Aquisition

In case of Real Time, the working methodology is base on creating graphical programming languages understandable for the processor and for the PFGA separately. For each processor being able to understand the code is necessary to write that code in an appropriate way, LabVIEW has Vis available for both kinds of application and trying to use one from FPGA in Real Time make produce some error when executing the project. In order to make possible the communication between both parts (Real Time target and FPGA) some communication methods must be used.

These methods depend on the model and which FPGA that is being used. The communication procedure requires that both parts are synchronized for sending and receiving data between both parts. This can be explained by continue thinking in an acquisition example.

If we think in the previous example, one of the parts acquires data from a sensor and writes it in a communication element. The function of the other part, normally the Real Time target is to read data from this communication element. If both elements do not read and write data with same or similar frequency, the communication element would be empty or overwritten and this will cause a crash in the system.

In the same way as working with xPC Targets based systems; cRIO is used in large different fields of applications in embedded control. In these different fields, there may be found applications from vision acquisition systems to exoskeleton control going

through high frequency signals treatments, encrypt data systems in digital format or traditional control systems.

The basic format of cRIO: the NI cRIO-9076 has an integrated processor integrates a RT processor with 400 MHz LX45 FPGA and has four module slots I / O C Series These input / output provide delivery capabilities and acquisition of analog , digital data and pulse trains of pulse width modulated (PWM) , CAN bus , LIN bus , Profibus , RS - 232 and RS -485 serial port timers ... there is a wide repertoire , and within each mode can be selected voltage ranges , exit / or unipolar differential input , opto ...

Parallelly, the myRIO is a controller that allows using the same concepts that have been explained in this section for cRIO but within a lower level. The myRIO is a device that has been design only for student use. The idea behind is to allow students getting used to these king of controlling devices and the use of the environment.

When working with myRIO you can use the SPI interface processor to access the peripheral input/output without using the FPGA , so can avoid long synthesis times FPGA . CompactRIO system for the communication interface with the PC uses Ethernet; however, the MyRIO system uses USB interface.

The errors made in the programming of the FPGA may be found at the stage of generation of VHDL code and synthesis. Contrary, LabVIEW cannot detect these errors before entering this phase.

4.4. LabVIEW environment

In this section I would try to introduce the main characteristics of software used. In order to explain this, first of all LabVIEW will be explain as the main concept and then it will be splitted into two main sections: Real Time and FPGA. After the general view and explanation of the most important concepts, there will be a section explaining the basic concepts of the LabVIEW project developed.

One of the improvements that are introduced in this project with respect to the one in which the robotic arm was design is the usage of a different controller. The myRIO, as it has been explained before, is a powerful device compared to an arduino. One of the advantages of this device is that it can be programmed using LabVIEW.

LabVIEW is a graphical programming language quite different from traditional programming languages such as C++ and Visual Basic. The first advantage that need to be mentioned is how intuitive is to create sophisticated and powerful programs with elegant, graphical user interfaces.

Historically, LabVIEW programs are called Virtual Instruments, or VIs, because their appearance and operation imitate physical instruments, such as oscilloscopes and multimeters.

One of the reasons for programming in LabVIEW is that, despite the fact that for being able to develop a project it is needed to be familiarized with this environment, anyone that has never programmed may be able to understand the code or at least, use it. This can be explained with various reasons:

1. Every project in LabVIEW usually has a user interface that allows the user interacting or making a program run. If this front panel is correctly documented any user may be able to run the VI.

2. LabVIEW is characterized by its modularity. Modularity can be understood as the degree to which a program is composed of discrete modules such that a change to one module has minimal impact on other modules. These modules are known as subVIs and its size may vary depending on the application.

3. LabVIEW projects can also include non-LabVIEW file types. For example, you can include documentation files. The idea is to explain the usage and the main characteristics of the project in this files and save everything together so that anyone that opens the project may read it and understand the code.

There are two different components in a VI. Each one has its own functionality and is structured in different ways.

4.4.1 Front Panel

The front panel is the user interface of the VI. As it has been explained, this must be the simplest part of the VI. The front panel of the main VI is the one that is known as the User Interface of the whole project. This one will be better explained in the corresponding part above.

In a front panel there are different elements that may be found:

1. Control: Input device. It may be knobs, buttons or slides. Its functionality is supply data to the block diagram.

2. Indicator: the opposite element is an output device that displays the data that is acquired or generated in the block diagram.

3. Toolbar: in LabVIEW there are two different toolbars depending if you are in the block diagram or in the front panel. The one that can be found in the front panel looks like this.

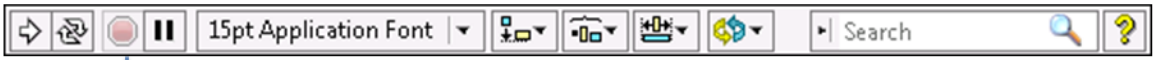

Figure 31. LabVIEW Front Panel Toolbar

This toolbar allows you to run, stop or pause the VI (control the program execution), reorder the elements present in the interface or search for an element.

4.4.2 Block Diagram

On the other side, the block diagram is where the main programming is located. In this section of the VI is where the designer decides which codes will be executed, in which order and in which device.

You build block diagrams by connecting the objects with wires. The color and symbol of each terminal indicate the data type of the corresponding control or indicator (shown in the front panel).

Similarly as it has been done before, types of elements that may be found here will be enumerated and briefly explained.

1. Constants: are terminals on the block diagram that supply fixed data values to the block diagram.

2. Terminals: data values you enter on the front panel controls enter the block diagram through the control terminals on the block diagram. During execution, the output data values from the block diagram pass from indicator terminals to the front panel indicators.
   They can be displayed as icon or not.

3. Nodes: are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. There are different nodes; they can be functions, subVIs or structures.
   Nodes are analogous to statements, operators, functions, and subroutines in text-based programming languages.

4. Wires: elements that transfer data between block diagram objects. They may indicate that the user is trying to connect terminals that are not compatible; if there is something wrong the wire appears as broken. It looks like a dashed black line with a red X in the middle.

Figure 32. LabVIEW Error Representation in Block Diagram

   Wires are different colors, styles, and thicknesses, depending on their data types

5. Tool bar: it has the same elements as the one in the front panel but also it allows the user to debug the VI. It has the buttons needed to debug the program: look for an error, see the execution order and how the program works (this functionality is not available in FPGA).

Figure 33. LabVIEW Block diagrm Toolbar

4.4.3. Dataflow

Now that there has been presented an overview about LabVIEW it is going to be explained the basic functionality elements.

One of the important things in this programming environment is that LabVIEW is based in Dataflow. This means that it does not use a control flow program execution model like Visual Basic, C++, JAVA, and most other text-based programming languages. In a control flow model, the sequential order of program elements determines the execution order of a program.

In LabVIEW, when a node executes, it produces output data and passes the data to the next node in the dataflow path. For this reason, the movement of data through the nodes determines the execution order of the VIs and functions on the block diagram.

For a dataflow programming example, consider a block diagram that subtract two numbers and then multiply by 2.00 from the result of the subtraction, as shown in Figure 35. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because the Multiplication function cannot execute until the Subtract function finishes executing and passes the data to the Subtract function.

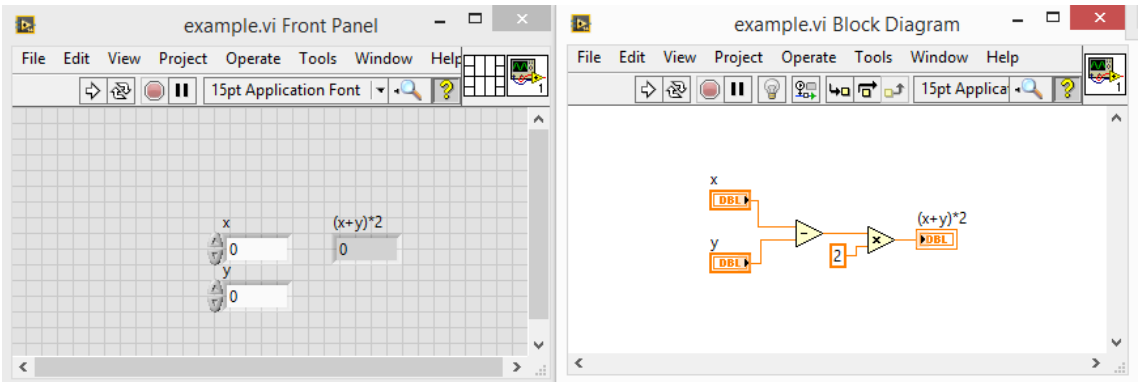

Figure 34. LabVIEW Example of Dataflow

Remember that a node executes only when data are available at all of its input terminals and supplies data to the output terminals only when the node finishes execution. In this case, this example code has been developed in order to demonstrate that no matters if there is a multiplication after a subtraction if, as in this case, the order of the calculations is fixed by code.

In a situation where one code segment must execute before another and no data dependency exists between the functions, other programming methods may be used in order to force the execution order, such as error clusters.

Chapter 5: LabVIEW Modules and Examples

Chapter 5: LabVIEW Modules and Examples

5.1 Real Time Target

After having a general view about what LabVIEW is and the basic concepts of a generic project the idea is to introduce both options able in myRIO to program. In this device, the user can choose whether to program in LabVIEW Real Time or in LabVIEW FPGA. In this section Real-Time concepts supported in this module of LabVIEW will be introduced.

Here, there are treated concepts such as what real time is oriented to programming, determinism, and jitter focus that have been already introduce in the Real Time section but more specifically explain for programming.

Here also myRIO Real-Time will be presented including the host and the target and which possibilities are included by now in LabVIEW.

For a review, Real Time means in-time. In other words, a Real-Time system ensures that responses occur in time, for this to happen, every operation in the system must be in time. One of the problems when using general purpose operating systems is that you cannot ensure that a response occurs within any given time period, and calculations might finish much later or earlier than you expect them to.

For a system to be considered real time, all parts of it must be real time. Therefore, if in an application where every part but one is working on real time, until this last element also works in real time this application will not be working in RT. This is the case of many applications in the world. In order to be clearer, an example may be given:

A critical test application would benefit from using a real-time system because the system must work reliably, possibly for an extended amount of time. If you think in a system that is used to measure and store some measurements in a hard disk, you will find three main elements:

1. Writer: this is considered the sensor element. In this case, this would be the part of the system where a sensor element take some measurements and write it on a communication element.

2. Communication element: in LabVIEW and precisely, in LabVIEW Real-Time (LVRT) there are different elements which functionality is communicate data between two elements, normally two different targets (e.g.: host and myRIO)
   For this example we only need to imagine this element as a way to send data between the writer and the reader.

3. Reader: its function is reading the data that has been already written by the writer in the communication element, and then save it in a hard disc.

Figure 35. Communication Diagram

A Real-Time application will write, communicate and Read in a reliable and fixed time. If one of these three elements does not work in real time, that is, has no reliability of finishing in a specific time the whole system will not Real Time.

5.1.1 Concepts related to Real-Time

For having a general but accurate view of this platform some concepts must be explained. In this part, some terms that apply to real-time applications will be enumerated with its correspondent brief explanation.

1. Deterministic Task: it is a task that must finish on time, always. Therefore, deterministic tasks need dedicated processor resources to ensure timely completion.

   For example, a control loop is a deterministic task because the control algorithm must complete within an expected time for the control algorithm to function correctly. An example of a non-deterministic task is logging data to a file because file I/O operations can occur with a lot of jitter. Deterministic tasks should execute at a high priority.

2. Priority: characteristic that defines when a VI or loop should execute relative to other VIs and loops.

   In LabVIEW, the execution order may be configured. What is more, correctly timed structures in RT programs should have a priority associated with them. This priority provides the RTOS with and order of importance when deciding what needs to be executed.

3. Loop Cycle Time: this may be described as the time required to execute one cycle of a loop.

   Many applications that require a real-time operating system, such as a control application, are cyclical. The time between the start of each cycle, T, is the loop cycle time, or sample period. 1/T is the loop rate or sample rate.

4. Jitter: is known as the variation of loop cycle time from the desired loop cycle time. In order to understand this feature better I will explain it with an image.

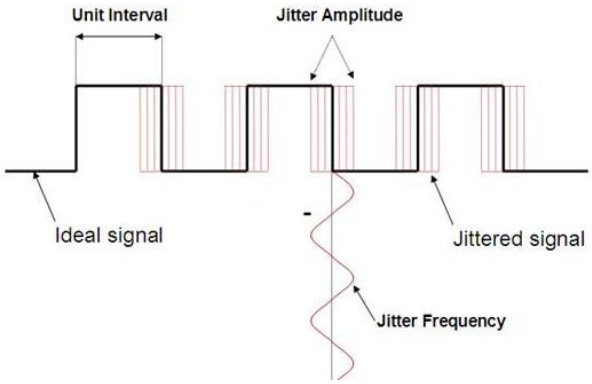

Figure 36. Jitter Description

In this Figure, it can be seen the variations that an ideal signal may suffer when working in a real system. It must be known that, even with Real-Time operating systems, the loop cycle time can vary between cycles. The maximum amount that a loop cycle time varies from the desired loop cycle time is the maximum jitter.

5. Determinism: is the measurement of jitter magnitude. It is used to indicate how reliably a system can respond to external events or perform operations within a given time limit.

High determinism, a characteristic of real-time systems, guarantees that your calculations and operations occur within a given time. Deterministic systems are predictable. This is important in a control application that measures inputs, makes calculations based on the inputs, and then outputs values that are a result of those calculations. Real-time systems can guarantee that the calculations finish on time, all of the time.

5.1.2 Module Overview

As it has been said, the objective of this chapter is to explain the main features of the two LabVIEW modules with which myRIO may be programmed. This explanation can be splitted in two classes: the main characteristics that are equal as the ones explained for LabVIEW full development system and the ones that are specific characteristics for Real Time Module.

5.1.2.1 Common characteristics with LabVIEW

There are some basic things that are characteristic from LabVIEW that are maintained in the module. Included in this group of features it can be found:

- The project is composed by Vis. These VIs are the place where the user code is located, this means that, in the same way as explained before, user can defined the functionality of the system with Vis.

- Every VI is composed by a front panel that may be used as user interface and the block diagram. For a user not familiarized with this environment, both block diagrams (The one belonging to LabVIEW and the one from Real Time Module)

may look likes the same one. However, as the reader will see in the following section the functions available in both, have some differences.

- Dataflow is maintained as the basis scheme of the environment.

- It is possible to create an application from the source code that allows the user to run the VI without having LV installed (it is only necessary to have a driver called LVRun-Time engine).

5.1.2.2 New features for Real Time Module

As well as there are new concepts when talking about Real Time environments. When programming this kind of systems and, in particular the controllers some additional functionality will be needed.

In this second group of features that are included in the module, there will be explained the new functions added for Real Time performance. Included in the characteristics of the module it can be found that:

- It is based on running the project in a target different from the computer.

- Inputs and Outputs from the target may be access from the project.

- There are new functions which objective is to monitor and control Real Time main features such as determinism.

- When programming, priorities may be defined and it is important to have them in mind when developing code.

5.1.3. Real Time project

In LabVIEW there are many devices that may be programmed with the National Instruments software. The way it works is the following: a device may need special features to be installed in the PC or controller in order to make the environment compatible with the device.

This concrete features may be even drivers, libraries, toolkits or a combination of some of them. In order to use the myRIO, despite the fact that it is a device from the same company it is necessary to install a toolkit.

By installing this toolkit, some functionality is automatically added to LabVIEW. In the Real Time part, it can be found this specific palette from which we can take advantage of some codes already done:

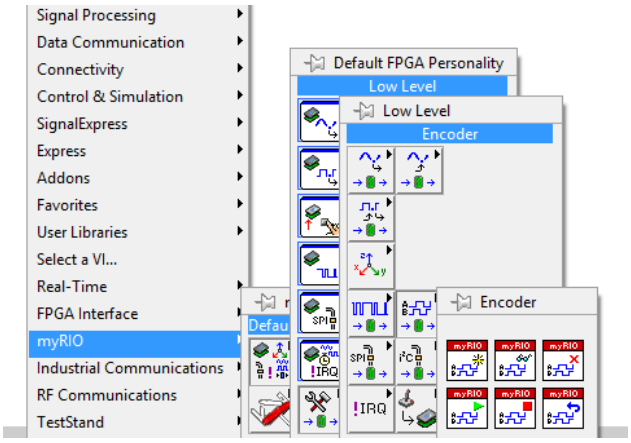

Figure 37. myRIO functions palette

As it may be seen in the Figure 38, there is already a PWM implemented among other things. For this reason, the example that has been implemented with this module may be seem less complex that the one from FPGA since the starting point was not an empty project.

In order to understand how do people work in LabVIEW environment, it will be illustrated the example of configuring the PWM VI. Imagine that the user wants to create a PWM control signal as in the robotic arm in order to control some actuator.

First, it is needed to open the myRIO palette and after finding the function desired, it will be dropped on the block diagram.

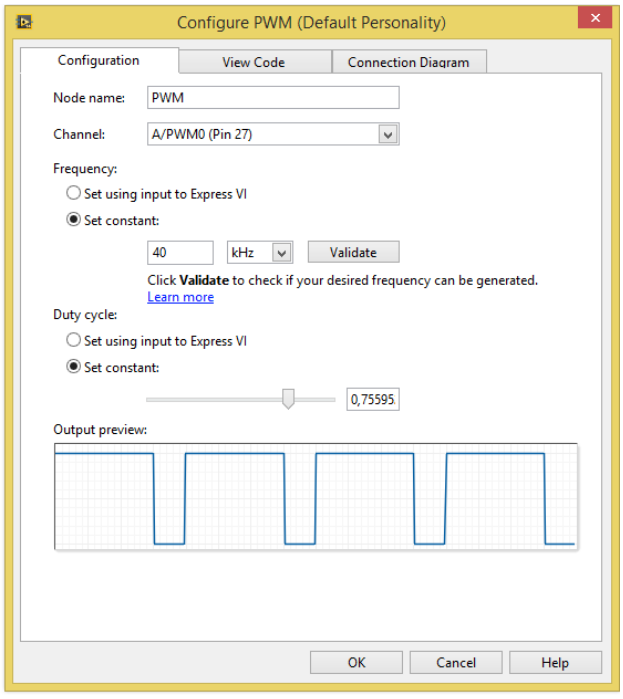

Figure 38. PWM Generation with Express functions

Afterwards, it must be configured for having the concrete specifications needed. In the Figure 39, it is shown how to select the pin in which the signal will be created, the frequency of the pulse and its size.

The advantage for using this method is that it is faster than building it from zero. Additionally, in this Vis you can access to the code so it may be included in a bigger project with the certainty that it works fine.

In this case, the code used to generate a PWM seems simple but it has subVIs that uses the FPGA resources and some complex structures. This is normally the main problem of using code that it is done, that you need to understand it completely in order to avoid execution problems.

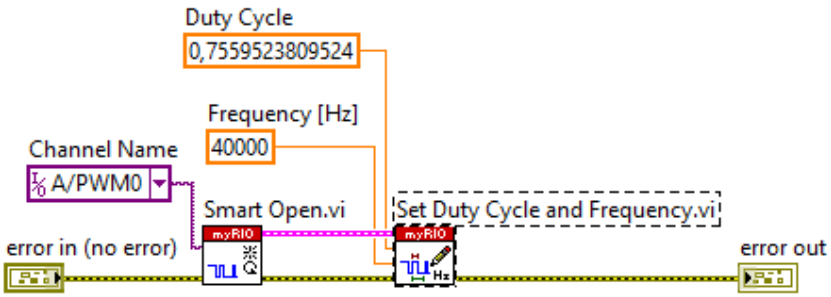

Figure 39. PWM Generation code with myRIO API

In the Figure 40 it is shown the higher lever of the PWM Generator VI.

5.2 FPGA

Before seeing an example of how would be programming and controlling a complete RT application, in this case a robotic arm in Real Time Module, it is necessary to describe the concept of FPGA and which part is developed by this chip. Before starting this explanation is good to remember which is the objective of this project in order to maintain the correct focus and correctly understand the whole work.

The aim of this work-study is to understand the concept of Real Time control and, particularly, go through two of the modules provided in LabVIEW environment to control a robotic arm. Now, that the reader has just been introduced in Real Time concept and seen one of the modules, it is time to described the most basic architecture when talking about programming.

The second and last module described from LabVIEW is FPGA programming. In this section, it would be explained not only the concept behind but also which function does the FPGA has in a complete myRIO application. Finally, there will be implemented an example of how would the arm be controlled using this LabVIEW module.

5.2.1 What means FPGA?

First of all, an FPGA corresponds to Field Programmable Gate Arrays. They are reprogrammable silicon chips with unconnected gates and other hardware resources. The idea of FPGAs is basically to program these gates in order to make the system works as desired.

The user can define the functionality of the FPGA by using software to configure the FPGA gates. There are several environments to do this, in this project it will be done using LabVIEW but normally it is done by programming the FPGA directly in VHDL.

5.2.1.1 VHDL

Equivalent to LabVIEW FPGA Model, the VHDL is a Hardware Description Language (HDL). The basis of these languages is to convert a design made in software to hardware by using logic gates. VHDL stand for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language.

.

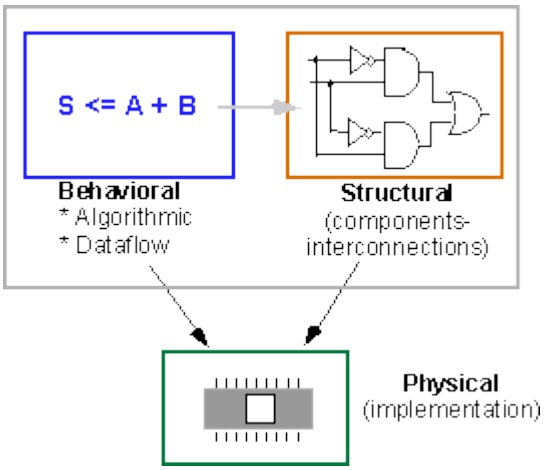

Figure 40. VHDL Representation

Every VHDL design description consists of at least one entity / architecture pair, or one entity with multiple architectures. The entity section of the HDL design is used to declare the I/O ports of the circuit, while the description code resides within architecture portion.

5.2.1.2 Concepts needed to understand FPGAs

It is important to understand the benefits of using FPGAs in spite of any other target or computer. The most important one is that these silicon chips are reconfigurable through software, making them very flexible.

Unlike a microprocessor or an application specific integrated circuit (ASIC), FPGAs are able to have their functionality modified even after being deployed in an application. Additionally, FPGAs applications are actually implemented in hardware without any software involved such as an operating system. Working with a chip that is

reconfigurable with changes in software is similarly to create new hardware anytime the project is compiled.

For these two reasons, FPGAs provide the ultimate in execution reliability and provide a powerful way to have a completely full hardware system working with different functionalities designed by the user.

Finally, as FPGA execution is controlled by the programmable interconnects, users can implement truly parallel tasks within the FPGA that will execute simultaneously and independent of one another. This parallelism is possible because there is no operating system on the module that must divide CPU time between several tasks.

In order to understand this feature let me give an example. Imagine that we need to calculate separately and at the same time two portions of code in the same chip. The way to do it is by using parallelism.

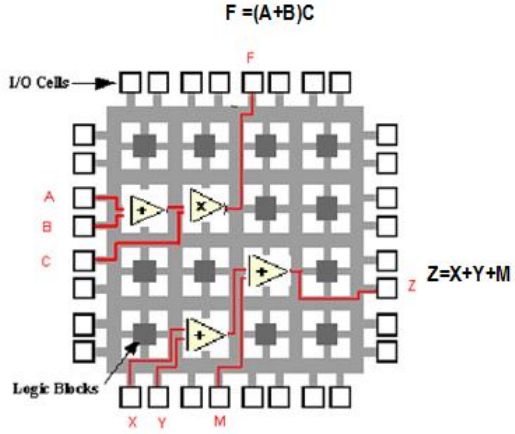

Figure 41. FPGA Parallel Executin Diagram

In this simple scheme can be seen how the FPGA will execute two different codes separately. In this example, the objective of this compilation of the silicon chip is to execute at the same time two operations: on one hand to calculate F and on the other hand, to calculate Z.

Taking into account that F= (A+B)*C and Z=X+Y+M.

The advantage of this feature is that it is possible to have two separate parts of codes in different execution times.

5.2.1.3 FPGA in myRIO

In the myRIO, the FPGA may be use on one hand to control the whole system with its correspondent programming configuration and without the need of RT target in the whole system. On the other hand, it may be used as an intermediate device in the data acquisition process.

In this second configuration, the FPGA makes possible to establish a data processing before being retrieved by the processor. This data processing only can be done by

using integers, fixed-point decimal numbers or just bits. It is important to know that when working with FPGA decimal floating point numbers are not supported, they can only be used if the FPGA has one or more co processing units. These units are not allowed in myRIO and low cost cRIO.

One of the main aspects that must be taken into account is the length expressed in bits of the data that is being processed. This data makes the number of logic cells available in the FPGA decrease very fast.

One of the limitations of FPGA is its processing capacity. This is restricted to a limited repertoire of operations, such as addition, multiplication, subtraction and low level operations, this peculiarities limits its scope.

Generally, they are systems that are very used by telecommunication engineers in applications like modulation and demodulation or script of digital signals. These operations are supported by these limitations and an FPGA is able to make them in nanoseconds. However, for complex operations, such as conditional executions, is better avoid the FPGA and programming a RT processor like the ones explained in the previous section.

In the following section an example of how controlling the arm with just programming the FPGA will be illustrated.

5.2.2. Module Overview

In general, execution in hardware is faster than execution in software. Since FPGA is a hardware solution, implementing algorithms on it has higher performance than implementing same algorithm in Software.

5.2.2.1 Common characteristics with LabVIEW

The advantage of programming the FPGA with LabVIEW Module is that there are some features, the main ones that are maintained with respect to the common environment.

As shown in the previous RT part, it is maintained the block diagram and the front panel, the graphical programming design, the data flow. But additionally it has some functionalities that are not allowed in order targets.

5.2.2.2 New features for FPGA Module

With FPGA you can create logic that can execute in one clock cycle, contrary to other programming habits, when developing code in FPGA it is important to think regarding to clock ticks. Although this may be also possible by using ASICs which generally have higher performance than FPGA, the development cost and flexibility are the tradeoffs. In this line, there exists an specific type of loop, only available for this module which is called single cycle time loop.

The single-cycle Timed Loop does not include enable chain registers inside the loop, which saves time and space. This means that LabVIEW automatically optimizes code inside a single-cycle Timed Loop to execute more quickly and consume less space on the FPGA, compared to the same code inside a While Loop.

In the following block diagram, the code within the While Loop takes four clock cycles to execute, excluding the overhead of the While Loop, which takes two additional clock cycles to execute. The red vertical lines indicate where each clock cycle ends inside the While Loop. The same operation in a single-cycle Timed Loop executes within one clock cycle, if the clock period is long enough.

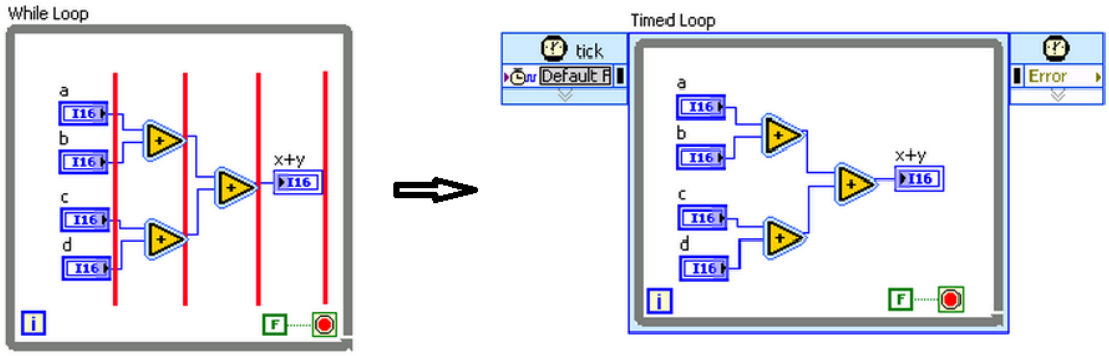

Figure 42. Time Loops

It is highly deterministic. With FPGA, you can implement code that synthesizes deterministic hardware down to the tens of nanoseconds. Moreover, it is a frequent habit to create different clocks based depending on which functionalities are desired.

For creating a derived clock it is necessary to know which based clock is included in the FPGA used. Then the user just need to fixed the new time base as a multiple of the other one.

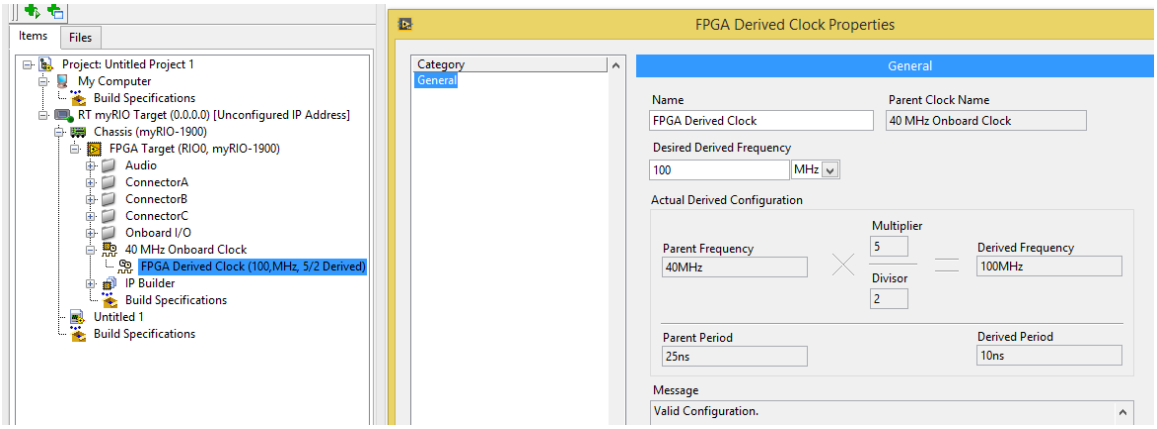

Figure 43. How to create a derived clock

In the Figure it is show how would a derived clock of 100MHz looks like in a myRIO project. In this case, the controller has a 40MHz clock by default so it is possible to work with derived clocks creating from it.

Offload processing:

- FPGAs can also be used to off-load computationally intensive processing so to free up the CPU for other tasks.

- When programming FPGA there is some needed time to compile the code and translate it to hardware. This step may long from 5 minutes to hours and is here when some errors may be found.

5.2.3 FPGA project

In this last part before explaining step by step how to build a complete application, it will be show a sample project created to be executed in the myRIO. The objective of this example is first, show how the arm may be control just using FPGA, familiarizing with this environment and also to understand which limitations it has.

As it can be seen the first limitation is that in FPGA it is also possible to access to the controls and indicators by doing a reference to the specific VI. For this reason, it is shown both, the FPGA VI and the RT one. It is important to have in mind that both Vis are running in the myRIO and there is nothing in the laptop as show in Figure 45.

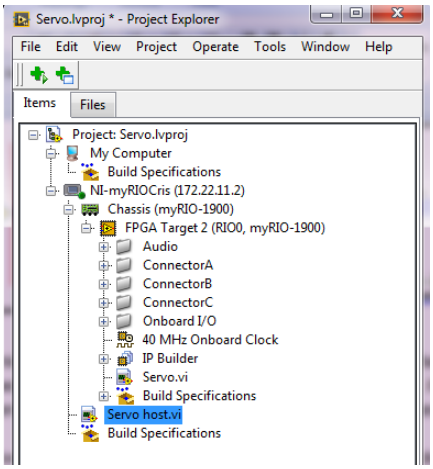

Figure 44. LabVIEW Project Appearance

In this Figure 46 is shown how a PWM can be generated in the FPGA manually. As it is a Digital Waveform which its only parameter is the duty cycle or pulse with the only thing needed is to fix the signal TRUE or FALSE depending on it. In this example, it will be possible to fix the signal period depending on the servo (not every device work with the same signal range). Afterwards, it is possible to vary the pulse with of each servo from the front panel in order to vary the position of each servo.

In the FPGA VI shown the PWM is generated. Then, it is necessary to send this position to the RT Target in order to access to the control that allows moving the servo when running the VI.

To access to the FPGA VI from RT there is a LabVIEW functionality called FPGA Reference VI. After selecting the correspondent VI it is used an Invoke Node of each parameter to vary the pulse with and Move the Servos.

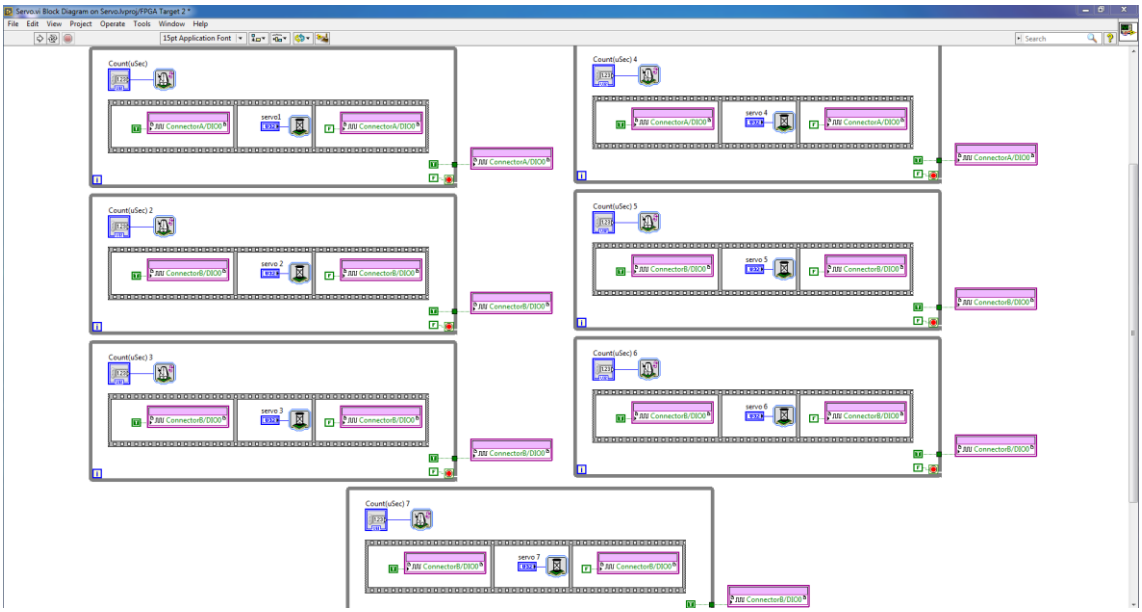

Figure 45. FPGA Block Diagram for controlling the Arm

To access to the FPGA VI from RT there is a LabVIEW functionality called FPGA Reference VI. After selecting the correspondent VI it is used an Invoke Node of each parameter to vary the pulse with and Move the Servos.

Finally, the Front panel may be personalizing. In this case the controls are created by following this manual and using Inskcape (previously explained):

https://decibel.ni.com/content/docs/DOC-4819

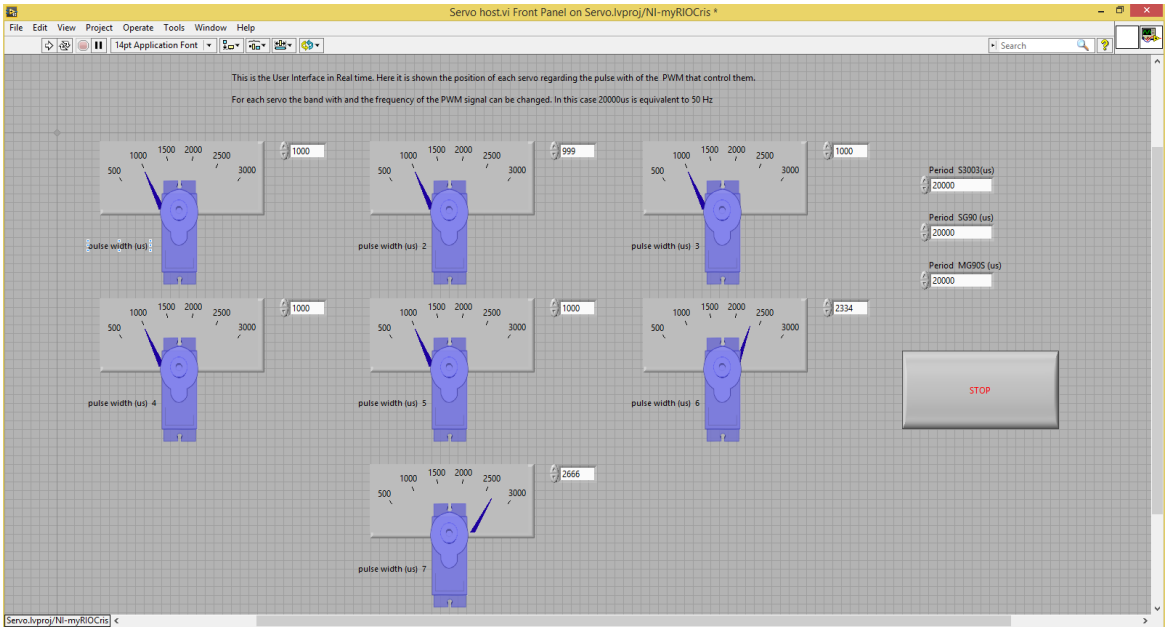

Figure 46. Real Time User Interface

5.3 How to create a RT application & Sample projects

The objective of this section is to explain step by step the approach of solving an ECM system (Embedded Control and Monitoring System). As shown in Figure 47, it uses a human machine interface (HMI) and hardware containing both a real-time processor and FPGA, where the FPGA is directly connected to modular I/O.

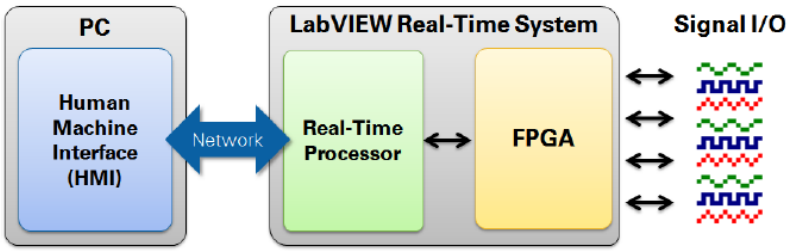

Figure 47. Embedded Control and Monitoring System

This architecture provides power and flexibility and it uses LabVIEW graphical programming environment to design the entire system. The steps needed to implement the hole systems are:

1. Configure hardware: in this case, as myRIO was design to be used by students, it is very easy to configure it. It is necessary to plug the usb cable and then follow the instructions given by the start up.

2. Identify application Requirements: in every application the first thing needed is to define which requirements are needed to fulfill the objectives.

   a. **Identify I/O and I/O Rate Requirements:** define the timing of each task in the real-time application by asking yourself the following questions:

- What type of I/O do you need to acquire from and generate to your application?
  In this case there will be used PWM output signals in order to control the position of the seven servos and Bluetooth input to read commands from the user to control the servos.

- At what rate(s) do you need to acquire and generate I/O?
  This fixes the reliability of the system. It can be varied if desired in the User interface and its limits are fixed by the characteristics of the components used.

   b. **Identify Processes:** before beginning coding, think through the high-level design of your application by defining the tasks the application must perform. In this case, there are two main processes:

- Read from Wiimote
- Send commands to move the correspondent servo

   c. **Identify Process Timing:** this means to define how much time it is needed to execute every part of the code correctly. In order to develop this step correctly the following questions should be answered.

- What causes the process to execute?
  Mainly it related the input to the outputs of the project. In this example, a command from the wiimote makes a servo move.

- What causes the process to sleep?
  In every application it is needed to define which event makes the whole system stop in order to make it efficient. In the sample application there is a stop bottom in the user interface

- How frequently does the process execute?
  In this part it is defined which processes are executed by events and which ones are periodic.

d. **Identify Data Transfer Types:** after you define the high-level processes that make up your application, define the data transfer relationships between those processes. There exist three main types of data transfer:

-Tag: transfer current value only and may have more than one writer and reader.

-Stream: transfers every value of a continuous stream of data and usually there is only one writer and one reader

-Message: transfers every value of intermittent data with low latency

In the Figure 49 is shown a complete diagram of the implemented application in which it is shown the processes executed in each part and the communication between each other.

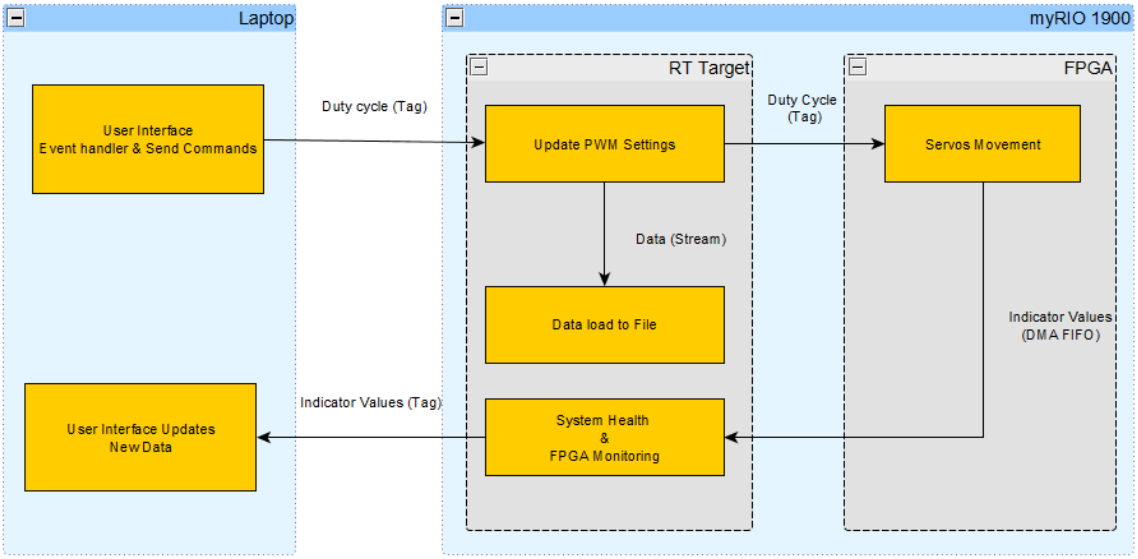

Figure 48. Whole Application Diagram

e. **Identify Performance Requirements**: Identify the requirements for the real-time application and tasks such as:

- Determinism: In the sample application there is only need determinism when moving the servos and when pressing stop bottom.

- Response time, CPU efficiency, Throughput…

f. **Identify Reliability Requirements:** Various real-time applications will require different levels of reliability. It is important for the user to determine the proper reliability requirements of your real-time application

3. Documenting your design: Before developing code it is important to create diagrams and documents (if needed) to explain how the application works, its parts and the features needed to allow other users to introduce improvements if needed or make some changes if sometime is needed.

4. Accessing the I/O: when using LabVIEW there are more than one way of accessing I/O depending on the device used. One of the advantages of using myRIO is that it is easy to access to any channel of the device.

   For accessing I/O in the myRIO it is needed to use the concrete Vis available in the myRIO palette. For example, if it were desired to access to the data from the accelerometer first it is needed to create a channel specifying the name, and then user can read and do the calculations desired with the data obtained.

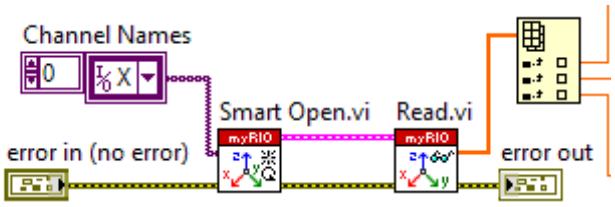

Figure 49. Accesing to accelerometer data from myRIO

   If working only with FPGA it is possible to pick the I/O module from the project and leave it on the block diagram as shown in the Figure 51:

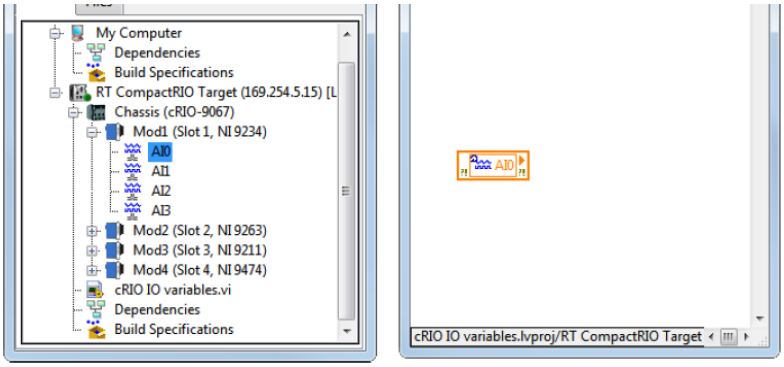

Figure 50. LabVIEW treatment to Target I/O

5. Programming FPGA: now that every configuration is defined and it is known how to access I/O is time to implement the code. In the case of FPGA it is already explained in the previous section but is important to know that in real projects the first thing to implement is the FPGA. Then it is gone to the upper level: Real Time Target and finally the host.

6. Programming Real Time: in case of the myRIO, there are many functionalities already implemented. However, is not always recommended to program this device using them because this RT functionalities may use some FPGA resources and if the FPGA is already programmed an error will be obtained.

In this sample project, the Real Time Target is used as intermediate between the host and the FPGA. In this part of the system data is transfer and it is possible to make some calculations if needed and some control if desired.

7. Create User Interface: the user interface is the element located in the host, in this case the laptop, which allows the user to understand and control the arm.

   In the sample application it is shown the two wiimotes used in order to monitor the commands is pressed and then the position of every servo. In this application there are two different UI, the second one is the User Interface in RT is the one shown in Figure 47.

Chapter 6: Conclusions

Chapter 6: Conclusions

6.1 Conclusions

To conclude, what does Real Time control really mean? I believe to help to put it in context Real Time control vs general-purpose control.

Many general purpose control systems run open loop. Meaning you run the controller with no death line. This type of control involves no determinism, but not only general-purpose control include this concept, but also there are many applications that may be done with no death line. Some examples may be save data to a file or move a piece from one point to another. In both cases the important thing is that at the end all task will be done, no matter how long does the system takes to do it.

Real Time control systems are typically closed loop systems where you have a tiny time window to gather data, process the data and then update the system. If you missed that time window, means you have a delay in some task, your system becomes unstable. That is not an option for application such as power conversion, data acquisition of High Speed I/O and advanced motor control in industrial processes.

So, firstly when developing a Real World industrial application, one of the main things to think about it is: Does the system really need to be deterministic? If so, which parts need to be executed in a concrete period of time and which do not have death time?

Afterwards, it is necessary to implement the whole system. While developing the complete code is necessary to think assign task to each element. To summarize this second conclusion, a brief summary will be done. So, what does it really take to perform a Real Time control application?

It is about thinking in the entire system. It starts with the I/O. Regarding the system, it has been said that Real Time Operating systems need fast, accurate integrated ADCs (Analog to Digital Converters) to keep up with the speed of the math engine core. These inputs directly communicate with the brain of the system which is the RT Target, in this case the micro controller of the myRIO.

Real Time Operating Systems require high performance that involves high efficient target to handle complex control algorithms. Then, there is the outputs of the system which in this case are PWM. These PWM should be smart and flexible to meet every possible applications regarding motor control.

Finally there is the mouth of the system, or communication with the external world, which is more important than ever before as customers need to bring more intelligence through networking. Furthermore, is not just about having this complete system components, is about having the right control architecture that integrates these components in order to meet the needs of the customer and that is what is not just about the silicon. It is necessary to provide complete Real World development tools and software design for an application from the ground up.

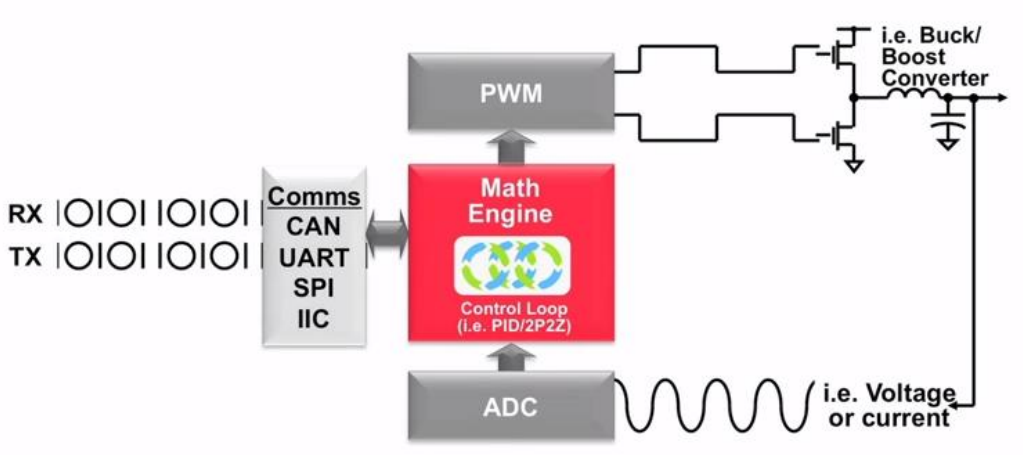

Figure 51. Real Time System Diagram

Concluding, the difficulty for developing a RT System is to correlate every part and learn how to manage the priorities of each task and also how to define which ones need to be deterministic. Additionally, the user should have to think about each task timing in order to avoid unexpected results or possible system crashes.

Finally, let's take a look of the example that has been developed to support the theoretical concepts. Although the control of a 3D printable robotic arm may be done by using general purpose control, in this case the objective was to learn how to define which task were deterministic and to differentiate every element that compose the application.

For the control of the robotic arm, firstly was defined that the Stop (emergency) and the generation of the PWM, this means motors movement, must be the deterministic tasks in this application. Furthermore, the Stop bottom will have the higher priority.

Additionally, the idea was to include also a non-deterministic task. For this, the wiimote was included in the project allowing the user to see one of the possibilities of how every element may be related.

The conclusion that I have taken by developing this example is that the most difficult thing is to take advantage of the characteristics of each part of the controller and also to manage timing in order to leave time to the processor to develop every task as desired.

At the end, it must be said that the comparison of both, the movement of one arm controlled by an Arduino and other controlled with myRIO has no sense. This is because as it is a project developed by student (with low budget) there is no sense in comparing the movement of servo motors that are no more expensive than 5€. In order to be such cheap, its precision is not as good as needed to compare the results of both executions.

Furthermore, as some servo models have changed it may be seen that both complete executions are very similar. However, in this project it is controlled with wiimote that

uses another protocol different from the one used for moving the seven servos, in this case the protocol used is ActiveX.

6.2 Improving suggestions

In this last section it will be enumerated and briefly explained the next implementations that may be included to continue this project.

There are two possible lines in which this process can be completed. On one hand it may be thought in use every concept explained in the report to build a RT application able to be used in industry.

In this case it will be necessary to use a complex controller, such as CompactRIO if maintaining the line of using LabVIEW and also a Real prototype. This second project needs to have a higher budget and better security functions.

On the other hand, it may be thought about including some more elements to this 3D robotic arm to make it more complex and robust. In this second case I would include:

1. Inverse kinematics: In this project, as it will be explained in the first annex, the kinematic base used is the forward one. In order to make the prototype moving only by knowing the end position of the arm and the base position, it is proposed to use the inverse kinematic.

   However, it is important to take into account that the calculations needed to control the arm with this kind of theory is more complex and needs more knowledge about mechanics.

2. Camera: it will be interesting to introduce a camera (compatible with the myRIO) and implement some code about detecting special elements or just to detect a possible dangerous element in the system.

   In the myRIO there is also an available Vision Palette that could make able to implement some ideas that complete the project.

3. Arm tool: this third possible improvement is oriented to create more that one project that allows using the prototype with more than one functionality such as designing a clamp in order to use it with the camera.

Annexes

ANNEX I. Kinematics

In this annex there will be seen a general review about kinematics and more precisely, the movement based on inverse kinematics. For this part of the project, there is already available a Final Degree Project that studies the kinematics of this particular robotic arm. The kinematic model that will be presented has been made by Carlos Rodriguez Zambrana in his final Degree Project: “Modelo cinematico y control de un brazo robótico imprimible”.

http://e-archivo.uc3m.es/handle/10016/20665

The idea of this subchapter is to first, introduce the main theoretical basis of kinematics and then, present a summary of the analysis made by Carlos Rodriguez about the relative movement of the extreme of the arm with respect to the base.

Theoretic Base Kinematic Features

Kinematics is a branch of mechanics that study of the motion of the bodies without reference to mass or force. For this issue there are used words, diagrams, numbers and equations. The goal of any study of kinematics is to develop sophisticated mental models that may be used to describe and explain the motion of real-world objects, in this case, the robotic arm.

In this chapter it will be assumed that there are some concepts known such as velocities, accelerations and momentums. From this point it is necessary to know that kinematics may be splitted in two main groups: inverse and direct kinematics. These two subgroups consist on solving two different problems in order to understand how the arm is moving. In serial linkage type robot, Forward and Inverse kinematics are methods to calculate co-ordinate of end-effectors and joint angles respectively. It is important to have in mind that they are opposite concepts.

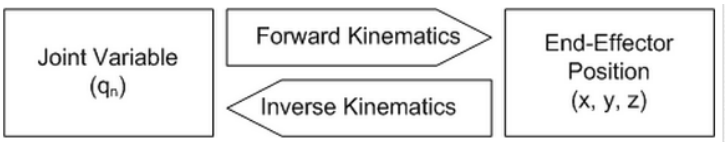

Figure 52. Inverse-Forward Kinematics Relationship

Inverse kinematics

The given inputs are the coordinates of the end-effectors, the outputs to calculate are the joint angles. The objective is to get all joints angle from given co-ordinate(s), path trajectory plan. This means that in this case, coordinates will be searched:

$$q = [q_1, q_2, q_3, q_4, q_5, q_6]^T$$

Equation 2. Structure coordinates

However, inverse kinematics could be tricky, for the same end-effectors coordinates, you may not have unique configuration, especially when the system is redundant. Dependently of the orientation of each joint, the extreme of the arm may being the same position but having different coordinate values. This kind of study gets complicated as the number of serial joints increases.
In order to solve this kinematic problem a solution like this equation must be found:

$$q_k = f(x, y, z, \alpha, \beta, \gamma) \text{ for every } k = [0,6]$$

Equation 3. Arm Position

In case of the robotic arm some restrictions must be done in order to allow any user to solved this problem. This can be explained by thinking about a concrete position and orientation of the arm and how much solutions exist that fulfills the equation above. This solution will be obtained by means of geometrical methods.

In order to simplify this study and taking into account that the three last DOF correspond to concurrent axes, the inverse kinematic study will be focus on calculate the values of the coordinates that fix the position that should take the arm. For this reason, the inverse kinematic problem is based on find out for which values of q_1, q_2 and q_3 the arm will be in the (x,y,z) position.

For this study, we know the length of each part, (L_1, L_2, L_3) and the position desired for the arm (px,p y,p z) and the objective is to find out enough equations that relates of q_1, q_2 and q_3 with this known data.

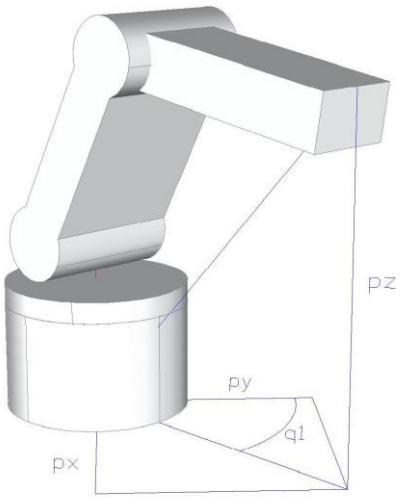

Figure 53. Calculating position

From Figure 53, by trigonometry it can be obtained q_1 applying the following equation:

$$q_1 = arctg\left(\frac{p_y}{p_x}\right)$$

Equation 4. q1 Calculation

For obtaining q_2 and q_3 we will be focus on the plane that is formed by parts 2 and 3 of the arm. By using the Cosine Theorem we have that:

$$\begin{cases} r^2 + {p_z}^2 = {L_2}^2 + {L_3}^2 + 2L_2L_3\cos(q_2) \\ r^2 = {p_x}^2 + {p_y}^2 \end{cases}$$

Equation 5. q2,q3 System

Solving for q_3:

$$\cos(q_3) = \frac{{p_x}^2 + {p_y}^2 + {p_z}^2 - {L_2}^2 - {L_3}^2}{2L_2L_3}$$

There will be two possible solutions for q_3 depending on the sign of the angle. These two possible solutions are represented in the Figure.

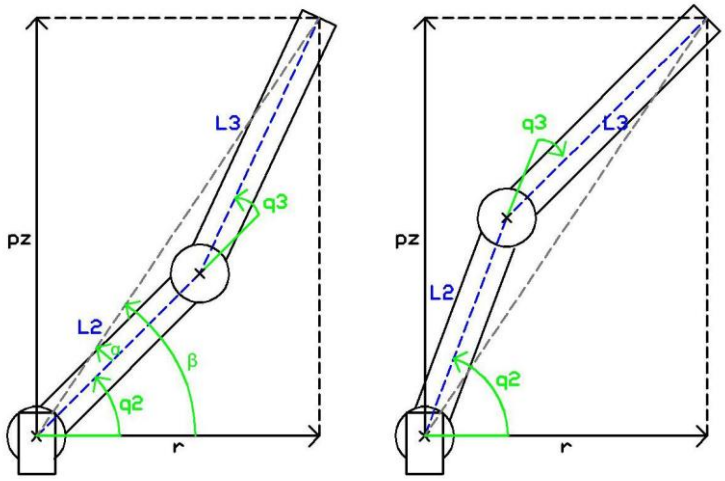

Figure 54. Angles diagram

This Figure also helps to visualize how q_2 will be obtained. In the picture of the left, it can be seen that:

$$q_2 = \beta - \alpha$$

Equation 6. q2 calculation

Being:

$$\beta = arctg\left(\frac{P_z}{r}\right) = arctg\left(\frac{P_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right)$$

Equation 7. Beta Calculation

$$\alpha = arctg\left(\frac{L_3 sen(q_3)}{L_2 + L_3 cos(q_3)}\right)$$

Equation 8. Alpha Calculation

There will be again two possible solutions depending on the sign that corresponds to the lower and higher position of the join.

Forward or Direct Kinematics

It is used to get co-ordinate of end effectors from given angles of all joints. This is used if it is known the joint angles and the lengths, then it is possible to compute the position and the orientation of the end effectors with respect to the base

For a multiple DOF robot, the forward kinematics is quite straightforward. In order to calculate the angles between axes it is used the Denavit-Hartenberg algorithm. The D-H parameters are associated with particular convention for attaching reference frames to each of the links that are present in the robot.

For calculating these parameters this steps must be followed:

1- Compute the link vector a_i and the link length. Considering that a_i is the perpendicular distance between the axex of the $joint_i$ and $joint_{i+1}$
2- Attach coordinate frames to the joint axes as shown in the Figure 55:

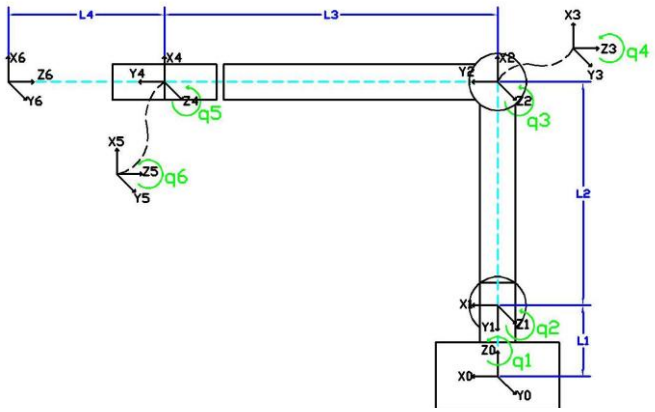

Figure 55. Coordinate Diagram

3- Compute the link twist α_i: define which is the value of the angle between the axes of $joint_i$ and $joint_{i+1}$ It is measured around x_i axis.
4- Compute the link offset d_i: is the distance between the origins of the coordinate frames attached to $joint_i$ and $joint_{i+1}$. It is measured along the axis of $joint_i$
5- Compute the joint angle φ_i: it is the angle around z_i axis between the link lengths α_{i-1} and α_i

join	φ_I	d_I	a_I	α_I
1	q1	L1	0	90
2	q2-90	0	L2	0
3	q3	0	0	-90
4	q4	-L3	0	90
5	q5	0	0	-90
6	q6	-L4	0	0

Table 2. Join parameters Results

6- Compute the transformation $^{(i-1)}T_i$ which transforms entities from $link_i$ to $link_{i-1}$. In order to compute this transformation matrix it must be applied this equation for each join:

$$i-1A_{i-1} = \begin{pmatrix} \cos(\varphi i) & -\cos(\alpha i).sin(\varphi i) & Sin(\alpha i).sin(\varphi i) & aicos(\varphi i) \\ \sin(\varphi i) & \cos(\alpha i)\cos(\varphi i) & -\sin(\alpha i).\cos(\varphi i) & aisin(\varphi i) \\ 0 & \sin(\alpha i) & \cos(\alpha i) & di \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Equation 9. Transformation (i-1)Ti

Finally, the Transformation Matrix can be calculated by:

$$T = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6 = {}^0A_6 \longrightarrow T = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & w \end{pmatrix}$$

Equation 10. Transformation Matrix

From the last row of this Transformation Matrix it can be obtain the position values that originally are searched:

$${}^0A_6 \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = (x, y, z, w) \rightarrow \vec{p} = (x, y, z)$$

Equation 11. Position values desired

ANNEX II. How to make the connection between LabVIEW and Wiimote

In this annex the objective is to explain how has been possible to control the robotic arm by using the Wiimote. This documentation has been taken from two different open web pages that will be summarized here.

https://decibel.ni.com/content/docs/DOC-1353

With a high-resolution camera, a 3-axis accelerometer, buttons, speaker, vibration feedback, and a Bluetooth wireless link, the Nintendo Wii Controller (Wiimote) offers exciting possibilities beyond use as a video game controller for the Nintendo Wii. With the examples in this document, you can easily access these data sources in NI LabVIEW.

The examples used for introducing this element in the project rely on a third party .NET Managed Library called “WiimoteLib” that allows to establish the link between the controller and the remote controller. This library is freely downloadable from the web. For this project I hav used the version 1.7 that is the stable one. However, there exist a beta version with Wii MotionPlus support, but it is buggy and incomplete and it's mostly unsupported at the moment.

The explanation about Wiimote will be divided in two different parts. On one hand, it will be briefly explained how to connect the wiimote with the PC and the main features of this kind of connection and how internally is done.

On the other hand, it is explained the LabVIEW part. This means that it will be briefly shown how has been included this components in the project and which kind of communication is used. In this part it is needed to take into account that the communication protocols are not compatible for every environment and programming tools. For this reason, this communication is not directly used in FPGA. However, this pint has allowed focusing the project in showing a complete RT project and not a one that only has FPGA.

General Features of Wiimote & First Steps

AS it has been explained in this first part the idea is to have a general view of the first steps needed to connect the Wiimote to the PC and the background of the communication that will be created.

How to connect the Wiimote to the PC

Because Wiimote uses Bluetooth to communicate with the Wii, it can be connected to and used by practically any Bluetooth capable device. This will likely be the biggest sticking point. It is important to mention that he Wiimote will not pair and communicate successfully with every Bluetooth device. In case your device is compatible the steps to follow in order to connect it are:

1. Start up your Bluetooth software and have it search for a device.

2. Hold down the 1 and 2 buttons on the Wiimote. You should see the LEDs at the bottom start flashing. Do not let go of these buttons until this procedure is complete. If pairing the Wii Fit Balance Board, open the battery cover on the underside of the balance board and hold the little red sync button.

3. Wiimote should show up in the list of devices found as Nintendo RVL-CNT-01. Balance Boards will show up as Nintendo RVL-WBC-01. If it's not there, start over and try again.

4. Click Next to move your way through the wizard. If at any point you are asked to enter a security code or PIN, leave the number blank or click Skip. Do not enter a number.

5. You may be asked which service to use from the Wiimote. Select the keyboard/mouse/HID service if prompted (you should only see one service available).

6. Finish the wizard.

At this point the LEDs at the bottom should continue to flash and you should see the device listed in your list of connected Bluetooth devices. On one hand, if you run the test application included with the source code and you see the numbers change, the device is already successfully connected. On the other and, if you don't see them change or you get an error, try the above again, but it is possible that the device is not compatible and it is not possible to connect to your PC.

Start communication with Wiimote

When the Wiimote is paired with the PC, it will be identified as a HID-compliant device. In order to connect to the device, it is necessary to use the HID and Device Management Win32 APIs.

As there is no built-in support for these APIs in the current .NET runtime, It will be used the P/Invoke. P/Invoke allows one to directly call methods of the Win32 API from .NET. The difficulty of this is finding the right method signatures and structure definitions which will properly marshal the data through to Win32. In this project it will be used part of code already design but if it is needed to change some parts, a great resource for these signatures is the P/Invoke wiki.

The process to begin communication with the Wiimote is as follows:

1. Get the GUID of the HID class defined by Windows
2. Get a handle to the list of all devices which are part of the HID class
3. Enumerate through those devices and get detailed information about each
4. Compare the Vendor ID and Product ID of each device to the known Wiimote's VID and PID
5. When found, create a FileStream to read/write to the device
6. Clean up the device list

Wiimote I/O and HID Reports

In the world of HID, data is sent and received as "reports". This means that it is a data buffer of a pre-defined length with a header that determines the report contained in the

buffer. The Wiimote will send and can receive various reports, all of which are 22 bytes in length, and all of which are explained at the links above.

When we obtain the FileStream on which to communicate with the Wiimote, we can start communication. Because reports will be sent and received almost constantly, it is essential that asynchronous I/O operations are used. In .NET, this is quite simple. The process is to start an asynchronous read operation and provide a callback method to be run when the buffer is full. When the callback is run, the data is handled and the process is repeated.

LabVIEW Environment

In this second section it will be explained some of the examples used to understand how the Wiimote communicates with LV. These examples that will be show are available on the Internet and they may be easily download from one of the links included in the Bibliography.

I will be mentioned some off the sample Vis available on this project. As it has been said this is the version 1.7, which has been chosen because of stability and reliability in this environment.

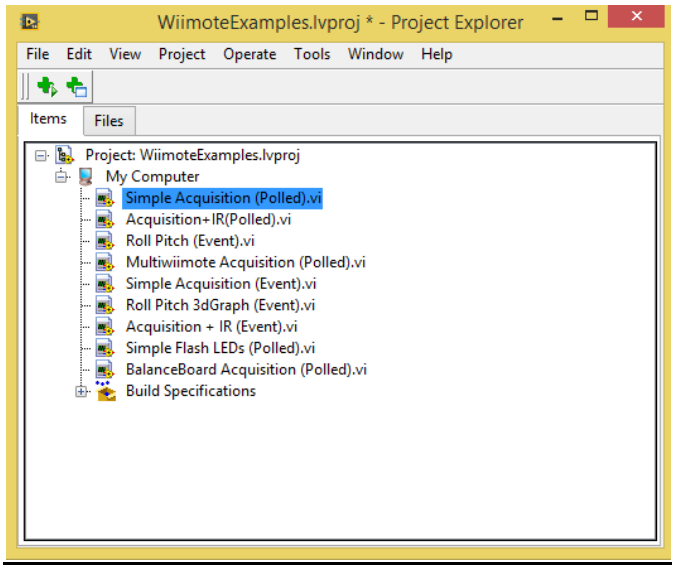

Figure 56. Wiimote Examples Project

First of all, it will be shown the simplest example in the project. The comments about the Figure 57 will be divided in two parts:

- Front Panel: Although it is not used in this project, this screen shot is shown in order to allow the reader to see that there are plenty of possibilities to customize the User Interface.

  In this case, it has been created a User Interface that is very similar to a Wiimote.

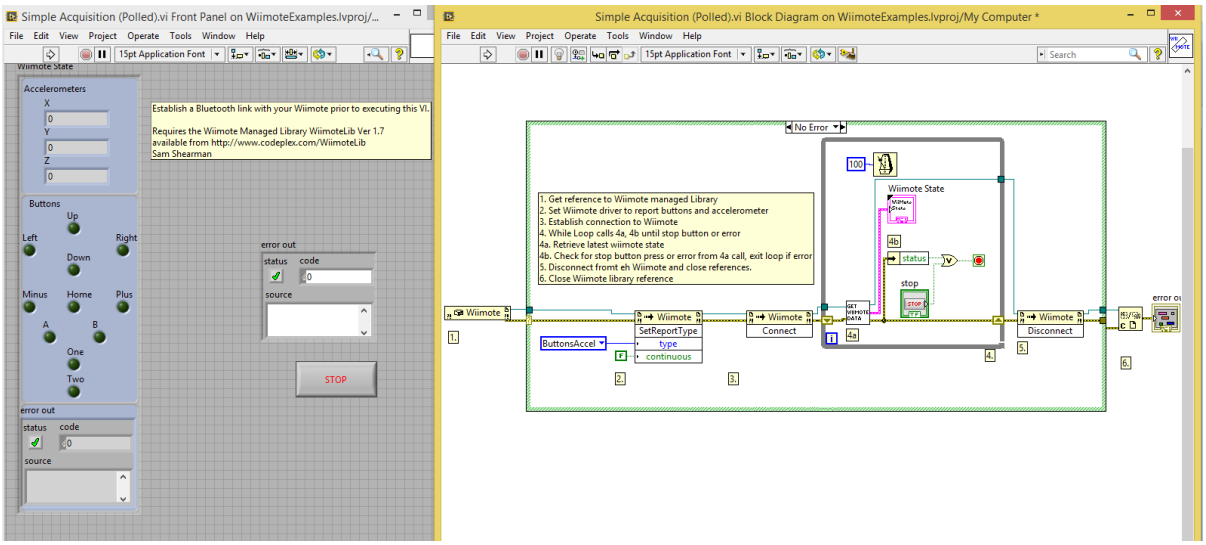

Figure 57. Wiimote VI

- Block Diagram: it can be seen that the protocol used for this is the .NET. Although I have not used this protocol before, the advantage of programming in LabVIEW is that every communication protocol has the same elements:

    a. Create the channel
    b. Open the communication
    c. Read
    d. Write
    e. Stop Task
    f. Clear
    g. Close

After having a general idea of how it is done, in order to end up this annex it will be show which elements would be added for use more than one Wiimote. In this project, there are used two of them.

In this case, there are some things that change with respect to the previous example. The first thing is how the communication is establish. As now there is more than one device it is needed to design the code to read from all of them.

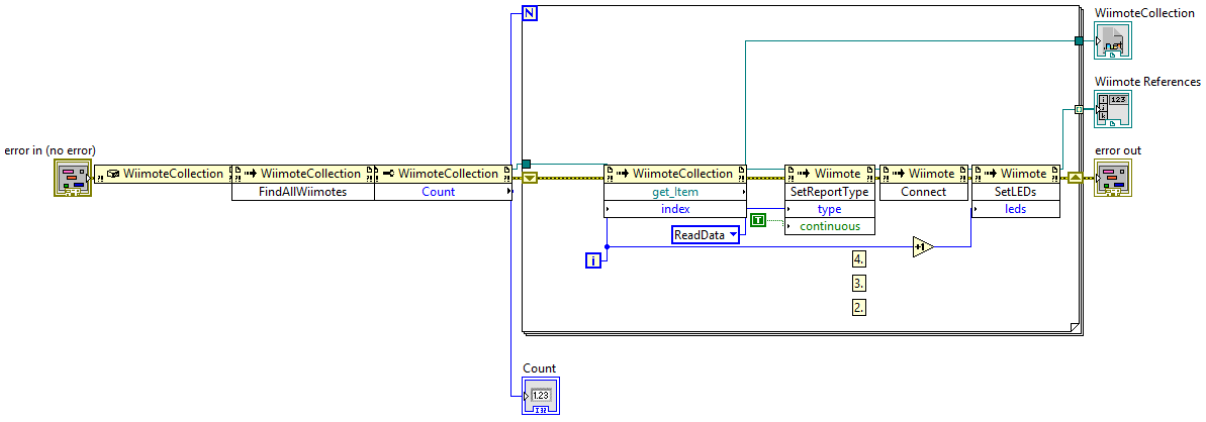

Figure 58. Multiple wiimotes block Diagram

As it is shown in the Figure 58, the first thing that changes is the way to access to more than one device. In this case, first is look for the number of elements used and then it is use a For loop to read one by one each Wiimote. After doing this, each device is set to connected state and in that moment they are prepared to be used.

In the example that I have used, the data obtained from the Wiimotes are shown in the front panel by using the Array shown in the Figure 59. When pressing stop bottom, the all communication busses are disconnected and clear.

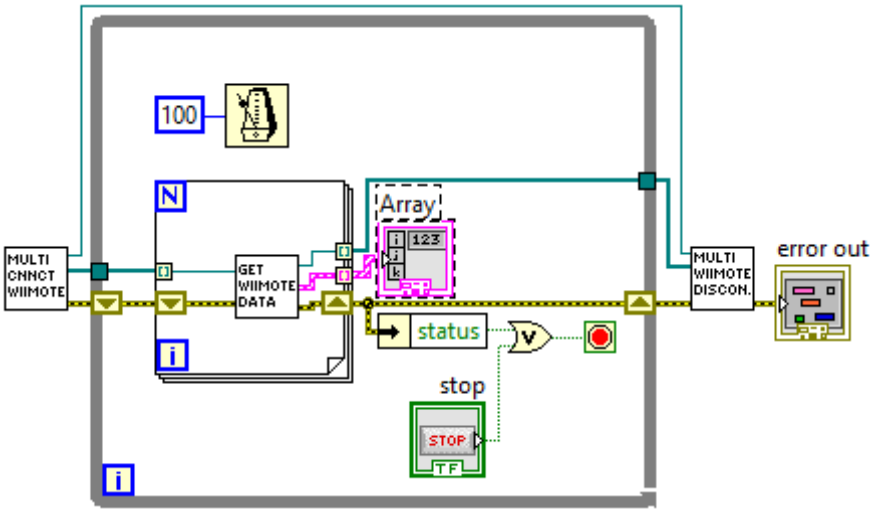

Figure 59. Clearing buffer and siconnect the wiimote in LabVIEW

However, in the project in spite of displaying the data from the devices in the front panel, the data obtained is send through a communication transfer element to make the proper control in the FPGA Target.

Finally, it is necessary to remember that although in LabVIEW it is allowed to use this kind of communication, in FPGA is not. For this reason, as it has been explained chapter 5, it is necessary to used some communication elements to send and read data from FPGA.

ANNEX III. BUDGET

Amount	Element	Price	Total Price
Controller			
1	myRIO for students + software	250,00 €	250,00 €
Motors			
1	Tower Pro MG995		0,00 €
3	Tower Pro SG90	4,75 €	14,25 €
3	Futaba S3003		0,00 €
Circuit & connection			
1	Perforated Plate	5,15 €	5,15 €
8,49 m	Cable	0,1573 €/m	1,34 €
Plastic used to build the arm			
0,03	ABS	40€	1,2€
0,03	PLA	50€	1,5
2m	Lined Heat shrink tube	1€	2€
Total Budget			**275,34 €**

Table 3. Budget material used in the project

Bibliography

Bibliography

Final Degree Projects and PhD

http://biblioteca.uc3m.es/uhtbin/cgisirsi/x/UC3M/0/5?
http://e-archivo.uc3m.es/handle/10016/20665
http://e-archivo.uc3m.es/handle/10016/16924

Datasheets

http://www.micropik.com/PDF/SG90Servo.pdf
http://www.servodatabase.com/servo/towerpro/mg995
http://www.servodatabase.com/servo/towerpro/mg90
http://www.servodatabase.com/servo/futaba/s3003
http://www.ni.com/pdf/manuals/376047a.pdf
http://www.trossenrobotics.com/dynamixel-xl-320-robot-actuator

Wiimote part

http://wiimotelib.codeplex.com/discussions
https://channel9.msdn.com/coding4fun/articles/Managed-Library-for-Nintendos-Wiimote
http://www.wiibrew.org/wiki/Index.php
http://www.wiibrew.org/wiki/Homebrew_setup

General Documentation

http://www3.uc3m.es/reina/Fichas/Idioma_1/223.14048.html
http://www.ni.com/getting-started/labview-basics/environment
http://www.ni.com/community/
http://www.ni.com/tutorial/5247/en/
https://decibel.ni.com/content/docs/DOC-4819

http://www.thingiverse.com/thing:30163
http://www.yworks.com/en/products/yfiles/yed/
http://www.thingiverse.com/
http://www.fanuc.eu/es/es
https://inkscape.org/es/